AFRL-RI-RS-TR-2012-228

MULTICORE ARCHITECTURES FOR MULTIPLE INDEPENDENT LEVELS OF SECURITY APPLICATIONS

SEPTEMBER 2012

FINAL TECHNICAL REPORT

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the 88th ABW, Wright-Patterson AFB Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2012-228 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/ S /

STEVEN T. JOHNS
Chief, Trusted Systems Branch
Computing & Communications Division

/ S /

RICHARD MICHALAK
Technical Advisor, Computing
& Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)* SEPTEMBER 2012	**2. REPORT TYPE** FINAL TECHNICAL REPORT	**3. DATES COVERED** *(From - To)* APR 2010 – MAR 2012

4. TITLE AND SUBTITLE
MULTICORE ARCHITECTURE FOR MULTIPLE INDEPENDENT LEVELS OF SECURITY (MILS) APPLICATIONS

5a. CONTRACT NUMBER
IN-HOUSE

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Jonathan Heiner, Lt. William Stanton, and Lt. Brandon Froberg

5d. PROJECT NUMBER
MILS

5e. TASK NUMBER
IN

5f. WORK UNIT NUMBER
HO

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/Information Directorate
Rome Research Site/RITA
525 Brooks Road
Rome NY 13441-4505

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/Information Directorate
Rome Research Site/RITA
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
N/A

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-RI-RS-TR-2012-228

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. PA# 88ABW-2012-4875
Date Cleared: 10 SEP 12

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The Multiple Independent Levels of Security (MILS) initiative is focused on providing a solution to the problem of securing information environments made up of multiple classification domains. We propose that multi-core architectures have the ability to bolster the MILS effort. However, current MILS operating systems are not designed for multi-core platforms. They do not have the hardware support to ensure that the separation kernel policies will be adhered to in a commercial off the shelf (COTS) multi-core environment. We propose to investigate the applicability of state of the art multi-core architectures to multi-level security through two means: 1. Develop low-level software code, e.g. a hypervisor, which can better enforce both time and space separation in commercial multi-core processors by putting individual cores in isolated states when processing sensitive information, such as policy enforcement decisions. 2. Design a new multi-core processor from the ground-up with MILS in mind. This processor will be designed with hardware features to help the kernel enforce the given policies.

15. SUBJECT TERMS
Multicore, Multiple Independent Levels of Security, Hypervisor, Virtualization, Cache Architecture

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** **JONATHAN HEINER**
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	SAR	74	**19b. TELEPONE NUMBER (*Include area code*)** **N/A**

TABLE OF CONTENTS

Section **Page**

LIST OF FIGURES

Figure **Page**

LIST OF TABLES

Table **Page**

ACKNOWLEDGEMENTS

Special acknowledgement to Captain Roy A. Porter, previously of the Advanced Computing Architectures Division of the Information Directorate within the U.S. Air Force Research Lab, for his contribution to this research. Capt. Porter initiated the project, acquired external funding, and provided the initial research towards the Multi-core MILS processor. Capt. Porter’s research in relevant cache hierarchies led to the selection of the Non-inclusive Cache Inclusive Directory (NCID) cache design that will be detailed further in the report.

Special acknowledgement to Richard P. McNally, of the C3I Division of the Defence Science and Technology Organisation within the Australian Government Department of Defence, for his contribution to this research. Mr. McNally’s early contributions to the effort helped guide the research in determining how to partition the physical memory by means of the Advanced Configuration and Power Interface (ACPI) memory structure provided by the Basic Input Output System (BIOS). This memory structure identifies usable and non-usable regions of the Random Access Memory (RAM). With this knowledge, the idea was to partition the memory regions of the RAM so that they are mapped to specific regions in the cache.

This project was funded by the Air Force Cryptographic Modernization Office in Lackland AFB, TX. Point of Contact (POC) is Mrs. Joyce Brookins.

1.0 SUMMARY

The Multiple Independent Levels of Security (MILS) initiative is focused on providing a solution to the problem of securing information environments made up of multiple classification domains. We propose that multi-core architectures have the ability to bolster the MILS effort. However, current MILS operating systems are not designed for multi-core platforms. They do not have the hardware support to ensure that the separation kernel policies will be adhered to in a commercial off the shelf (COTS) multi-core environment. We propose to investigate the applicability of state of the art multi-core architectures to multi-level security through two means:

1. Develop low-level software code, e.g. a hypervisor, which can enforce both time and space separation in commercial multi-core processors by putting individual cores in isolated states when processing sensitive information, such as policy enforcement decisions.
2. Design a new multi-core processor from the ground-up with MILS in mind. This processor will be designed with hardware features to help the kernel enforce the given policies.

2.0 INTRODUCTION

Mission requirements currently rely heavily on the difficult task of handling, processing and communicating information at various classification levels and access policies. This is especially true in the Joint environment where oftentimes there is the need to stringently control information that is shared with other nations. However, the current model of equipping users with wholly separate systems, one for each classification network, is costly in terms of IT resource utilization (especially for airborne assets) and more importantly, is inefficient at supporting necessary and approved communication. For this reason, initiatives are currently underway to develop architectures such as MILS. The goal of MILS is to effectively automate the access control of information with far fewer resource requirements, through the enforcement of application separation and controlled sharing.

In addition, the same techniques of separation and controlled sharing can be applied to the larger concept of improved computer security. This means that the MILS approach is capable of both increasing the availability of information at different security classification levels while increasing the overall security of the computing system.

Due to the difficulty of enforcing controlled sharing policies at the hardware level, current MILS separation kernels (SK) are not designed for operation on multi-core platforms. In order for MILS operating systems to be effective, i.e. trusted, they must be small enough in structure to be fully evaluated. Full evaluation becomes exceedingly difficult when hardware uses shared resources. These shared resources can give rise to both overt and covert channels. While it may be possible to add capabilities to the SK to enable the control of such low-level collaboration, such changes will likely expand the size past a verifiable limit. The following two sections provide an introduction to MILS and multi-core processors.

2.1 MILS Architectures

Multiple Independent Levels of Security (MILS), originally proposed by John Rushby in 1981 [1], uses a separation kernel to govern and provide system resource separation between non-kernel partitions. For our intent and purpose, we identify non-kernel partitions as guest virtual machines (VM) that are governed by a separation kernel executing on system resources.

MILS applications are based on four simple security policies: data isolation, control of information flow, periods processing, and fault isolation. Data isolation provides a guarantee that the data of one VM cannot be accessed by another VM. The control of information flow guarantees that intra-VM communication can only occur through authorized paths. Periods processing guarantees that there is no leakage of information covertly or overtly. Fault isolation guarantees that the compromise of one VM does not allow the compromise of another VM.

Effective Multiple Levels of Security architectures have been a topic of research and development for several decades due to the ever-present problem of users with heterogeneous information access rights working together. Although ideally this would not be a problem assuming users strictly adhere to their clearance restrictions, we know that the problem of unauthorized dissemination does occur. This is especially true when the computing base on which that information is stored and processed cannot be fully trusted and is a source of leakage itself. Thus, Boettcher et al. establish that the issues of policy enforcement and shared resources must be examined in combination [2]. In its present configuration, MILS uses a separation kernel and virtualized operating system (OS) to control the access to various shared system resources, enforce policy and allow trusted and untrusted applications to operate in the same environment [3]. Figure 1 shows an example of the MILS architecture.

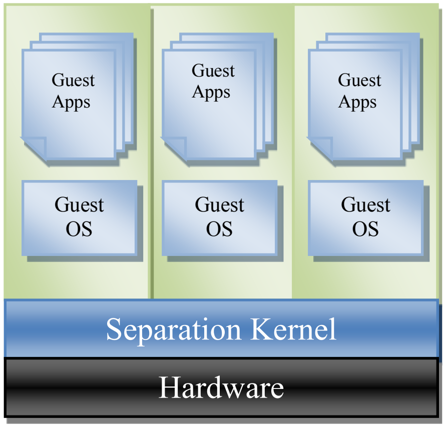

Figure 1: MILS Architecture

Research into effective MILS architectures is being given renewed interest by academic, commercial, and government entities. Dr. Alves-Foss et al. of the University of Idaho currently researches MILS applications to increase OS security [4]. Green Hills, LynuxWorks, and Wind-River all have MILS compliant separation kernels for single-core processors [5, 6, 7]. Mark Vanfleet from the National Security Agency recognizes the advantages of MILS architectures for the development of High-Assurance Embedded Computing [8]. Additionally, Huffmire et al. implemented a separation mechanism on reconfigurable hardware that provided separation of two soft-core processors, an Advanced Encryption Standard (AES) Cryptographic Engine, and associated Dynamic Random Access Memory (DRAM) [9]. This research shows that it is possible to use a reference monitor to isolate and separate resources in both time and space on a multi-processor system.

However, none of the published research provides a solution to cache coherency in multi-core architectures for MILS applications. Our research focuses on developing techniques in both hardware and software to guarantee separation of information in processor architectures that utilize shared cache resources.

2.2 Multi-core Architectures

Multi-core processors are designed such that two or more processing cores reside on the same physical chip and are capable of cross communication. On many multi-core processors, each internal core is identical in design to a single core processor except for the sharing of certain memory structures between all the cores (see Figure 2). This structure allows each core to process concurrent tasks from different applications, or portions of a single task in parallel. In the latter case, one core assembles the results from all parallel cores [10].

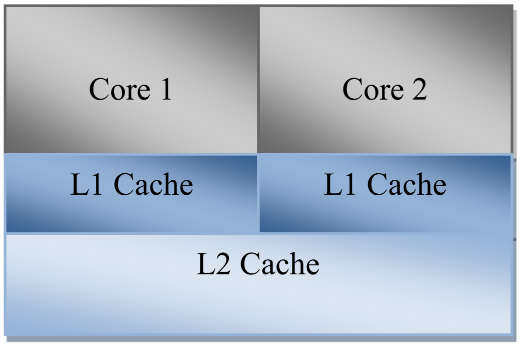

Figure 2: Generic Dual-core Processor

While multi-cores have initially been geared towards addressing the issue of disproportionate resource utilization to performance ratio, which is increasing in traditional chip design, the technology offers benefits to various applications, including multi-level security.

Multi-core processors present unique difficulties when using virtualization [2]. Separation kernels provide the lowest level of communication, and control over resources, through resource scheduling. However, current multi-core processors implement their own level of resource sharing, specifically in data storage. Multi-core architectures also implement a level of communication beneath the separation kernel. Current separation kernels do not provide the fine-grained control over these lower level system resources that is necessary to guarantee absolute isolation in time and space. While it may be possible to build this control into the kernel, it is likely that such additions would increase the size and complexity beyond that which is easily evaluated as trusted.

An example of the shared resource problem with multi-core processors is the last level cache (LLC). Since the LLC is a shared resource, it is necessary to strictly enforce all access rights. The current virtualized MILS approach addresses a similar issue through the separation kernel. The kernel arbitrates an application's access to data and thus ensures that an application cannot request information that it has no clearance for. The issue of a core's process accessing higher levels of data is slightly more complicated because these requests occur at a level outside of the kernel's control.

Modern processors have support for virtualization extensions. Advances in virtualization extensions allow for the hardware-based separation of virtual machine memory spaces. The processor will be able to maintain different page tables for individual virtual machines. In this manner, when a virtual machine tries to access another virtual machine's memory space, it will not succeed since it only has access to its own memory space. The only limiting factor is that current MILS platforms control processes while these virtualization extensions operate on virtual machines.

2.2.1 Covert Channels. Percival [11] illustrates the threat that shared memory caches pose to systems that process sensitive information e.g. cryptographic applications. The threat of cache analysis that exists for hyper-threading on uni-core systems is also applicable to simultaneous execution on multi-core systems because of the shared LLC. This threat is the primary concern for this research.

2.2.2 Overt Channels. Authorized communication avenues exist for multi-core systems that are not necessarily applicable to uni-cores. Of concern here is that if the system is not configured properly to disable these avenues, or the system is compromised, then the separation that exists at the application level is nonexistent once the information resides in hardware.

2.2.3 Authorized Communication. The issue of authorized information sharing across different levels is also not yet clearly addressed by the current MILS approach and may be suitable for a multi-core based solution. Boettcher et al. state that MILS is geared towards handling several processes at different classification levels at any given time, but with little cross communication [2]. It is a significant resource burden, especially in military environments, to handle information from several different levels to compile a hybrid creation. This is an important aspect of multi-level security and one can argue, even more pressing in terms of risk avoidance [12, 13].

3.0 METHODS, ASSUMPTIONS, AND PROCEDURES

Our research seeks to resolve the issues of implementing MILS applications on multi-core architectures. In particular, we seek to eliminate the shared properties of the LLC. There are two general approaches to eliminating the shared properties of the LLC. The first approach is to not use COTS processors and instead redesign the cache hierarchy in such a way as to selectively disable sharing within the LLC. This approach is embodied in our first task: to design and develop a multi-core MILS processor. The second approach uses current COTS processors and implements a software based solution in the form of a hypervisor to control where the data resides in the cache. This approach is embodied in our second task: to design and develop a multi-core MILS hypervisor. The following sections respectively detail both tasks.

3.1 Task 1: Multi-core MILS Processor

The Multi-core MILS Processor (MMP) task seeks to combine the architectural advantages of multi-core processors and the security features of MILS systems. This combination of features will generate a new architecture for multi-core processors that is inherently compliant with MILS applications. As previously mentioned, Huffmire et al. provides an example of using an arbiter between processors, co-processors, and memory to govern and provide separation of information on the system [9]. This task's original approach was to build a similar MILS arbiter between cores to govern and provide separation of information between the cores and caches. However, after further discussions with stakeholders the approach was modified to target cache hierarchies that have the potential to provide separation in the cache. Capt. Porter identified a method called NCID (non-inclusive cache, inclusive directory) [14]. In addition to the normal lines in a cache, NCID proposes the use of lines that contain only tags and not data. This is done for performance reasons, but it can also be leveraged to allow for data that is private to a particular core to be stored in LLC. We focused on creating a framework and associated concepts for the support hardware to attain separation through modifications to NCID.

Capt. Porter leveraged current Intel research on NCID to create an architecture that keeps all data that is not to be shared exclusively in the lower level cache. The lower level cache is private to each core and therefore, certain data that exists within lower level cache will not be visible or directly accessible to other cores. The NCID architecture defines an associated tag directory which is inclusive for all tags but not data. For the MILS architecture, access control bits are added to the directory lines, one for each core. When a socket or core sends a request for a specific line, the associated access control bit is checked to see if it is authorized to receive that data. Additionally, Capt. Porter researched a modification in which data is allowed to bypass the LLC and go directly between off-chip memory and the lower level cache. This technique is useful when the data should only be visible to a single core. Bypassing the LLC guarantees that other cores will not have access to the core's data, because even the data's tag will not be included in the LLC.

Lt. Stanton and Lt. Froberg replaced Capt. Porter as principal investigators for this task. To continue the research, Lt. Froberg identified the following subtasks:

1. Exchange of information pertaining to the MMP effort
2. Identify platform for testing
3. Continue NICD research/implementation
4. Collaboration with University of Idaho (UI)

3.1.1 Exchange of information pertaining to the MMP effort. The NCID concept, shown in [14], originally seeks to improve the performance of cache snoops by reducing the amount of information stored in a cache, thus reducing the number of snoop checks. An inclusive cache requires the LLC to include in its memory all the data currently residing in the lower level caches. On multi-core processors where the lower level cache is private per core and the LLC cache is shared, an inclusive shared LLC cache will eliminate the need for snoops as the data already exists in LLC. This reduces the snoop time but suffers space inefficiencies due to duplicated data and decreased flexibility. Non-inclusive caches require greater snoop times since all levels of the cache need to be checked among all the cores but saves space by reducing duplication. Exclusive caches guarantee that any data in the lower level cache is not replicated in the shared LLC. The NCID architecture decouples the data and tag management within the cache. Data in the LLC is not required to be inclusive while all the tags are, meaning that cache lines referencing privileged data will only contain the associated tags while the data is masked.

Capt. Porter's overall objective was to implement NCID into an existing multi-core architecture to meet the objectives of the MMP. [14] seemingly shows how NCID can be leveraged for efficiencies within a multi-core processor. One example usage could be to leverage the NCID for secure processing in differentiating security levels. The NCID would be leveraged to store different classifications in different portions of the cache. The OS would tag data either classified or unclassified and the hardware mechanism would force classified data to bypass the LLC and store the data in the private lower level cache, thus only the tags of the data are stored in the NCID. Only unclassified data can be stored with its tag in the NCID[1]. An example of such an implementation can be seen in Figure 3. The cache space in this example is divided into two separate sections, one for non-privileged data which includes both tag and data entries and the privileged section only containing tag values. This scheme does not require any modification to the actual HDL implementation of the cache, and is therefore non-invasive. Several cores would operate at varying privilege levels of security, whereas more privileged cores could access data tagged at equal or lesser privilege levels within the lower level cache of other cores. Lesser privileged cores are not allowed to access higher privileged tagged data within the lower level cache of other cores.

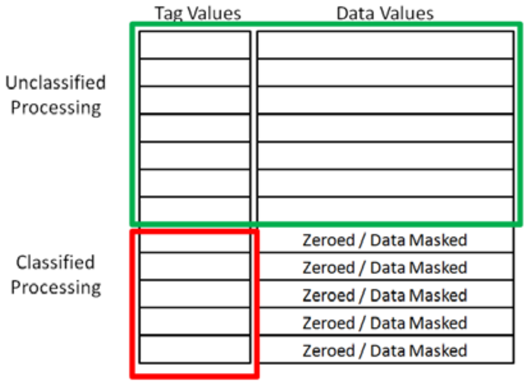

Figure 3: NCID application for classified processing

3.1.2 Identify platform for testing. To test and design the MMP, we used an FPGA development board. FPGAs provide a flexible programming architecture where the design can be quickly prototyped prior to fabrication. For our platform, we chose to use the Xilinx ML505 Xilinx University Program (XUP) FPGA development board. We chose this board for two

[1] Unclassified data can also be stored in the lower level cache and thus only have tag in the NCID.

reasons: 1. The ease in integrating the chosen OpenSPARC multi-core soft processor, and 2. The University of Idaho was using the same board for a contracted parallel line of research that supported this project.

One of the first decisions was to figure out where we were going to make the modifications to the existing OpenSPARC processor for integration of the NCID. The source code for the OpenSPARC T1 and T2 are available in Verilog [15]. These processors can be built with existing software development tools. However, the OpenSPARC T1 has a Xilinx Integrate Software Environment (ISE) project prebuilt that uses the Xilinx MicroBlaze module that leverages C code. Having familiarity with C over Verilog, it was decided to modify the C code of the MicroBlaze to perform the functionality of the NCID.

It should be noted that the original attempt to integrate the NCID into the MMP was intended with an OpenSPARC T2 since it was the latest/newest OpenSPARC processor. However, once the code-base was changed to C the T2 was not capable of integrating the C code-base. Thus, the MMP NCID implementation was retooled for the OpenSPARC T1. A major supporting factor for the change was the University of Idaho's complementary efforts in using the T1.

3.1.3 Continue NCID design/implementation. The best source for understanding the NCID architecture is the paper by Zhao et al. [14]. The NCID whitepaper presents various recommendations for implementation. Two specific implementation variations are the statically and dynamically addressed NCID. Ultimately, they recommend performing a simple addition, or modification, of the cache to support what appears to be a data masking mechanism. The mechanism would only allow the tags associated with the memory to be stored in the NCID. The data portion of the cache line would be 'masked' or omitted in the NCID but would be present in the lower level cache.

The method chosen was to model the simplistic, static NCID modification as seen in Figure 4.

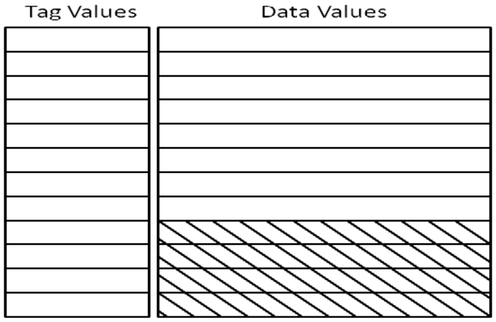

Figure 4: NCID Static implementation

This method does not extend the cache, nor change the addressing architecture of the OpenSPARC T1 processor. The processor would need to be modified only to provide the data masking mechanism. The first step was determining where the cache is implemented in the OpenSPARC T1 source code.

As mentioned previously, the NCID implementation needed to first determine what portions of the OpenSPARC T1 source code contained cache code. Both the C and the Verilog code can be accessed, updated, and compiled for use via the Xilinx Embedded Design Kit (EDK) tool. It should be noted that Xilinx EDK version 11 was leveraged on a Red Hat x64 OS, and version 13 was used on a Windows 7 machine.

We conducted an investigation into the implementation of the Cache Crossbar (CCX), which is used to control cache inquiries and traffic between the OpenSPARC cores. After reviewing the cache description contained within the OpenSPARC T1 Micro Architecture Specification [16], we decided to start looking for the Verilog code for the CCX. However, no Verilog source was found for the CCX within the OpenSPARC T1 FPGA source files. We eventually found that the CCX itself is not implemented in Verilog, but rather emulated using a MicroBlaze processor. Fisher et al. stated that an HDL solution was not feasible due to the overall size and resources required to implement a coherent lower level cache, CCX, and relevant interconnect paths [17]. Therefore, the Xilinx MicroBlaze core was implemented by Sun Microsystems to provide a smaller CCX solution to process all cache memory transactions. Figure 5 is a block diagram relating to how the MicroBlaze core serves as the CCX, and the flow of cache instructions and data between the T1 core(s) and main memory.

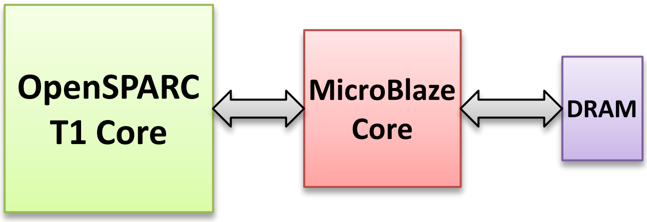

Figure 5: System block diagram for OpenSPARC T1

The our strategy at this point was to go into the CCX firmware implemented on the MicroBlaze processor, study it, and modify it to meet the requirements of NCID. Unfortunately, progress on this task was limited for the last 3 months due to software compatibility and licensing issues with Xilinx ISE. Progress should rapidly advance in the future once these licensing issues are resolved.

3.2 Task 2: Multicore MILS Hypervisor

A hypervisor is a low-level extremely lightweight reference monitor that resides within local memory, also known as a Virtual Machine Monitor (VMM). There are two types of hypervisors: Type 1, known as a bare-metal hypervisor, runs directly on the systems hardware. Bare-metal hypervisors manage hardware resources and guest software. Type 2 hypervisors, known as hosted hypervisors, runs on an already installed OS. Both Type 1 and 2 hypervisors provide the system with hardware virtualization capabilities.

Hardware virtualization also comes in three models: full, partial, or para- virtualization. Full virtualization requires a hypervisor to completely simulate the host hardware, allowing for the execution of a non-modified guest OS. Partial virtualization requires a hypervisor to simulate some of the hardware resources, requiring some modification of the guest OS. A hypervisor enabling paravirtualization provides no simulation of the hardware resources and requires extensive modification of the guest OS.

Our goal is to provide a low level hypervisor that:

- Runs directly on the hardware - Type 1
- Requires no modification of the guest OS - Full virtualization
- Provides separation of data - MILS

In order to accomplish this goal, we divided task 2 into three subtasks: 1. Analyze Intel's virtualization extensions, 2. Modify a Type-1 multi-core hypervisor for MILS, and 3. Test/Implement the Multi-core MILS hypervisor. Figure 6 shows the hierarchy of a system employing the Multi-core MILS Hypervisor.

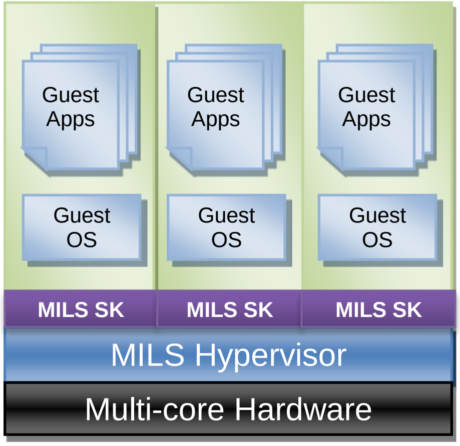

Figure 6: Multi-core MILS Concept Diagram

3.2.1 Intel Virtualization Extensions. The VMM's instruction set architecture (ISA) uses additional instructions to create, separate, monitor, and manage several virtual machines. These additional instructions are known as Intel's virtualization extensions. The first task is to understand these extensions and identify a means to partition the cache space between the processors cores. Our primary source of information on Intel's virtualization technology comes from the Intel 64 and IA-32 Architectures Software Developer's Manual, volume 3B [18]. By understanding the virtualization extensions, we are able to understand how hypervisors (VMMs) manage a specific processor's resources.

Intel's virtualization extensions are specific to Intel processors and vary depending on the processor family and model. Our primary qualifications with respect to choosing the target processor were the multi-core capabilities and implementation of the latest Intel hardware virtualization extensions. At the time of the start of this research, late 2009, Intel's most modern processor was the Core i7 processor based on the Nehalem architecture. This processor supports both VT-X and VT-D technologies. VT-X provides enhanced hardware virtualization extensions and VT-D provides input/output (IO) virtualization extensions. The Intel Core i7-860 processor also sports four physical x86 cores that can be split into 8 logical cores with simultaneous multithreading (SMT), dedicated L1 and L2 caches and an 8 MB shared L3 cache. When SMT is used, the L1 and L2 are shared among the two logical cores of a physical core. To reduce the amount of sharing involved in the cache, we chose to disable SMT. We chose to target an Intel Core i7-860-based system for the development of our multi-core MILS hypervisor due to its number of cores, large cache, and virtualization support.

A VMM needs a security policy to ensure that the separate VMs are unable to interact, modify, or affect one another. Intel virtualization extensions use a privilege-based security policy. The highest privilege entity, the VMM, has full control of the host hardware resources and provides the lower privilege entities, guest software, with abstractions of a virtual processor and allows the guest software to execute on the hardware while maintaining selective control of the resources [18]. The guest software operates at a reduced privilege level and functions independently of other virtual machines. Each virtual machine provides its own software stack so that it acts as if no VMM is present. This privilege base policy is maintained in hardware through privilege level bits attached to various hardware resources. Forced hardware checks of these privilege level bits allow or disallow access to hardware resources. Intel's privilege-based security policy is shown in Figure 7, where higher privileges are lower numerical values and lower privileges (less access) are higher numerical values.

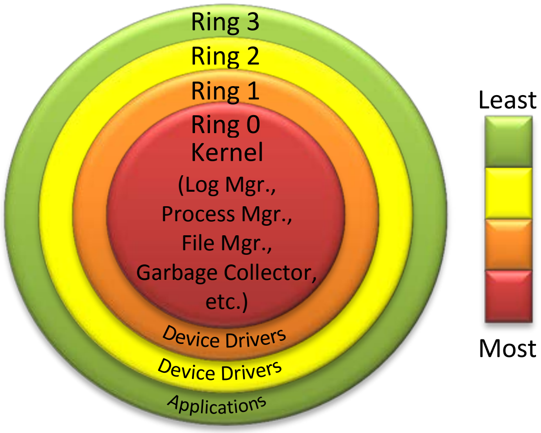

Figure 7: Intel's Ring-based Security Policy

Further analysis of the documentation provided instructional steps for creating, managing, and destroying VMs. Additionally the documentation provided insight into allowable operations within the VM. Unfortunately, we were unable to find any specific virtual machine extension (VMX) instruction that would help to partition the cache among the available cores. Nevertheless, the understanding of virtualization gained through this process was invaluable for the development of the Multi-core MILS Hypervisor.

3.2.2 Modify/Test/Implement Type-1 Multi-core Hypervisor for MILS [2]**.** To design the Multi-core MILS hypervisor, we decided to not build a hypervisor from scratch as that is not the intent of this research. For this purpose, we contracted with the University of Idaho to evaluate, choose, and/or modify a current existing hypervisor. Their choice was the Tiny Virtual Machine Monitor (TVMM) hypervisor developed by Kaneda [19]. The TVMM only supported AMD single-core processors, so UI modified the hypervisor to support Intel single/multi-core processors and renamed it the Intel-based hardware-Assisted Virtual Machine Monitor (IAVMM). The first version, received in the Fall of 2010, only supported single core operations. However, the final multi-core version of the hypervisor was delivered in the summer of 2011. The final version does not support the execution of full guest OSs, and only provided visual output for 2 of the 4 cores.

As shown in [11], the use of hypervisors is not enough to prevent an attack. The goal of this task was to modify the IAVMM hypervisor provided by UI to partition the cache so that cache memory pertaining to one VM residing on one core could not overwrite another core's VM cache memory. Partitioning physical memory is easy enough since we can directly address its memory locations. However, there is no way to directly address cache memory locations.

Since the L1 and L2 are dedicated to each core, we concentrated solely on the L3. The i7-860 L3 is an 8MB, 16-way set associative, 64 B line, inclusive cache. During a cache miss the processor will automatically load 64 B from physical memory to a corresponding set in the L3 and the L1 or L2. Based on the set-associativity of the cache, addresses from physical memory are hardwired to specific sets in the cache. The only means of partitioning the cache so that we can control where each VMs data resides in the cache is to partition the physical memory.

There are several means of partitioning the physical memory. Mr. Richard McNally, a visiting scientist from the Australian Government who was working on the project at the time, suggested using the system address map interface provided by the ACPI [20]. The BIOS provides a structure in memory that conveys a map of the memory resources, known as the e820 map, to the bootloader (VMM/OS). The thought was to use the VMM to modify the e820 map to convey our partitioned memory per VM. Upon further reading, the structure referenced by ACPI is actually known as the multiboot information structure [21]. Table 1 displays the contents of this structure. The left column represents the memory offset value of each field, the center column represents the field type, and the right column represents the present condition based on the flags field.

Table 1: Multiboot Information Structure

Offset	Field	Present Condition
0	Flags	Required
4 8	mem_lower mem_upper	Flags[0] set
12	boot_device	Flags[1] set
16	cmdline	Flags[2] set

[2] Even though the Modify Type-1 Multi-core Hypervisor for MILS task and the Test/Implement Multi-core MILS Hypervisor task are two separate tasks, they have been combined in this report for ease in reading since several iterations were performed between modifying and testing.

20	mods_count	Flags[3] set
24	mods_addr	
28-40	syms	Flags[4] or Flags[5] set
44	mmap_length	Flags[6] set
48	mmap_addr	
52	drives_length	Flags[7] set
56	drives_addr	
60	config_table	Flags[8] set
64	boot_loader_name	Flags[9] set
68	apm_table	Flags[10] set
72	vbe_control_info	Flags[11] set
76	vbe_mode_info	
80	vbe_mode	
82	vbe_interface_seg	
84	vbe_interface_off	
86	vbe_interface_len	

The E820 memory map is provided by traversing a memory buffer pointed by offset 48 (mmap_addr). The size of the memory buffer is provided by offset 44 (mmap_length). The memory buffer will contain one or more size/structure pairs. The size will indicate the size of the subsequent structure, and the structure indicates each memory range and type. Table 2 displays the contents of the size/structure pair. Base_addr is the starting address of the memory range where length is its size in bytes, and type indicates whether the memory is available or not. Table 3 displays the types of memory available. The map is guaranteed to list all standard physical memory available for normal use [21].

Table 2: Memory Map Size/Structure Pair

Offset	Field
-4	Size
0	Base_addr
8	Length
16	Type

Table 3: Memory Map Range Types

Memory Range Type	Description
Usable	This range is available RAM usable by the OS.
Reserved	This range is reserved by the system and cannot be included in allocatable memory pool by OS.
ACPI Data	This range is available RAM usable only after the OS reads the ACPI tables.
ACPI NVS	This range is reserved by the system for ACPI services and not to be used by OS.
Unusable	This range contains errors and cannot be used by OS.
Disabled	This range has not been enabled and cannot be used by OS.

Once we understood where the memory map was located and how to access it, we still needed to determine how to partition the physical memory among the cores so that all physical memory pertaining to a specific core will be mapped to a specific region in the cache. To accomplish this, we needed to know which physical memory addresses are mapped to the same set in the cache. Therefore, we needed to know how many cache lines and sets are in the cache. Equation 1 determines the number of cache lines in the LLC by dividing the total size of the cache by the size of each cache line.

$$\frac{8\text{ MB}}{64\text{ B line}}=128\text{K lines} \tag{1}$$

We can determine the number of sets in the cache by dividing the number of cache lines by the number of lines in a set, as shown in Equation 2.

$$\frac{128\text{K lines}}{16\text{ lines per set}}=8\text{K sets} \tag{2}$$

Since we have four cores, using Equation 3, we can divide the number of sets by the number of cores to identify how many sets we can allocate per core.

$$\frac{8\text{K sets}}{4\text{ cores}}=2\text{K sets per core} \tag{3}$$

Because a cache line fill will take 64 B of memory and physical memory is byte addressable, 64 addresses are mapped to the same cache line. We multiply the number of sets per core by the size of a cache line to determine the range of addresses assigned per partition. We assign the sets consecutively to allow for the largest possible partition size. Table 4 shows the initial mapping of addresses to sets and from sets to cores. Bits 17 and 18 of an address are used to map an address to a particular core. This results in 128 KB partition sizes. Given a 4 GB physical memory, we need to partition the physical memory into 32K partitions. This many partitions later proves to be a limitation to this technique which we'll discuss later.

Table 4: Initial Address Partitioning to Cores

Physical Memory Addresses (hex)	Cache Set	Core
00000000-0000003F	0	0
00000040-0000007F	1	0
…	…	…
0001FF80-0001FFBF	2046	0
0001FFC0-0001FFFF	2047	0
00020000-0003FFFF	2048-4095	1
00040000-0005FFFF	4096-6143	2
00060000-0007FFFF	6144-8191	3
00080000-0009FFFF	0-2047	0
000A0000-000BFFFF	2048-4095	1
000C0000-000DFFFF	4096-6143	2

000E0000-000EFFFF	6144-8191	3
…	…	…
FFF80000-FFF9FFFF	0-2047	0
FFFA0000-FFFBFFFF	2048-4095	1
FFFC0000-FFFDFFFF	4096-6143	2
FFFE0000-FFFFFFFF	6144-8191	3

The hypervisor uses the e820 map to identify how much RAM is available for system memory allocation and at which address ranges. From the map, they created a bitmap to identify free and allocated physical memory pages. We modified how the hypervisor reads the e820 map to include an additional memory map range type, and to partition only usable memory. The new range type was labeled as Reserved2 and indicated available RAM locations not available to the current core due to our partitioning scheme. As each core would execute this code separately, we compared bits 17 and 18 of the address to the CPU Identification (CPUID) of the executing core. If they matched, the addresses belong to a partition for the core; if not, they belong to a partition of another core. The modified code is shown in Appendix A.

Curiously, the e820 map provided by the system showed usable RAM from addresses 100000000h to 130000000h. This indicated an additional 768 MB of available RAM above the 4 GB range. Although the DRAM is marked as 4 GB of RAM we actually had 4.768 GB of RAM. The page tables and allocation bitmap provided by the IAVMM did not account for the extra memory and ignored it. This limitation was not fixed until later in the project and will be discussed later. Additionally, the memory map provided by the system included two ranges of addresses not labeled by any range type. Table 5 shows the original memory map provided by the system.

Table 5: Original e820 Memory Map

Start Address (hex)	Length (hex)	End Address (hex)	Range Type
0	8F000	8EFFF	Usable
8F000	1000	8FFFF	Reserved
90000	10000	9FFFF	Reserved
E0000[3]	20000	FFFFF	Reserved
100000	CCD60000	CCE5FFFF	Usable
CCE60000	110000	CCF6FFFF	ACPI NVS
CCF70000	26BE000	CF62DFFF	Usable
CF62E000	7000	CF634FFF	Reserved
CF635000	3A000	CF66EFFF	Usable
CF66F000	50000	CF6BEFFF	Reserved
CF6BF000	9C000	CF75AFFF	Usable
CF75B000	64000	CF7BEFFF	ACPI NVS
CF7BF000	26000	CF7E4FFF	Usable
CF7E5000	A000	CF7EEFFF	ACPI Data

[3] Notice a gap of about 256 KB not assigned a memory type. Assumed reserved memory type.

CF7EF000	1000	CF7EFFFF	Usable
CF7F0000	F000	CF7FEFFF	ACPI Data
CF7FF000	1000	CF7FFFFF	Usable
CF800000	800000	CFFFFFFF	Reserved
F8000000[4]	8000000	FFFFFFFF	Reserved
100000000	30000000	12FFFFFFF	Usable

A modified memory map[5] using partitioned memory ranges is shown in Table 6[6]. The memory ranges are significantly smaller and there are now Reserved2 memory types.

Table 6: Core 0 Partitioned Memory Map

Start Address (hex)	Length (hex)	End Address (hex)	Range Type
0	20000	1FFFF	Usable
20000	60000	7FFFF	Reserved2
80000	F000	8EFFF	Usable
8F000	1000	8FFFF	Reserved
90000	10000	9FFFF	Reserved
E0000	20000	FFFFF	Reserved
100000	20000	11FFFF	Usable
120000	60000	17FFFF	Reserved2
180000	20000	19FFFF	Usable
1A0000	60000	1FFFFF	Reserved2
200000	20000	21FFFF	Usable
…	…	…	…
CCE00000	20000	CCE1FFFF	Usable
CCE20000	40000	CCE5FFFF	Reserved2
CCE60000	110000	CCF6FFFF	ACPI NVS
CCF70000	10000	CCF7FFFF	Reserved2
CCF80000	20000	CCF9FFFF	Usable
…	…	…	…
CF600000	20000	CF61FFFF	Usable
CF620000	E000	CF62DFFF	Reserved2
CF62E000	7000	CF634FFF	Reserved
CF635000	3A000	CF66EFFF	Reserved2
CF66F000	50000	CF6BEFFF	Reserved
CF6BF000	41000	CF6FFFFF	Reserved2
CF700000	20000	CF71FFFF	Usable
CF720000	3B000	CF75AFFF	Reserved2
CF75B000	64000	CF7BEFFF	ACPI NVS

[4] Notice a gap of about 640 MB not assigned a memory type. Assumed reserved memory type.

[5] Does not show full memory map but a sample.

[6] Only shows memory map for Core 0, Cores 1-3 will show different start and end addresses.

CF7BF000	26000	CF7E4FFF	Reserved2
CF7E5000	A000	CF7EEFFF	ACPI Data
CF7EF000	1000	CF7EFFFF	Reserved2
CF7F0000	F000	CF7FEFFF	ACPI Data
CF7FF000	1000	CF7FFFFF	Reserved2
CF800000	800000	CFFFFFFF	Reserved
F8000000	8000000	FFFFFFFF	Reserved
…	…	…	…

Partitioning the main memory created several unique issues. First, there are certain memory regions that cannot be partitioned. Usable memory below 1 MB cannot be partitioned. Also, usable memory between CCE60000H and 100000000H also cannot be partitioned. Each time usable memory in these ranges is partitioned an APIC failure occurs, rebooting the system. Therefore these regions are labeled reserved instead of usable.

Secondly, since the partitions are only 128 KB in size, any data structure needs to fit within 128 KB of memory. 37K[7] partitions equate to about 1 million pages, and if the bitmap holds a bit per page, we need 144 KB of memory for the bitmap. This is larger than 128 KB available per partition. Even if we ignore the extra 768 MB of memory, we still need 131 KB of memory for the bitmap. We tried to only include pages of partitions that belong to the core in the bitmap for each core, but this caused conversion problems when allocating and freeing pages because the bitmap to page values were no longer one-to-one. We successfully were able to overcome the conversion problem only to run into page faults upon entering the virtual machine (guest software). A page fault will occur when the address maps to a page that is not available.

Upon further investigation, we realized that the page fault was occurring because the VMM code was placed in 16 MB of consecutive memory locations within the RAM by the bootloader, Grand Unified Bootloader (GRUB). This meant that the VMM itself was breaking its partitioning scheme. The VMM code also included the virtual page tables used before jumping from real-mode to protected-mode. Whenever core 1 would boot, it would try to execute its software that actually resided within core 0's VMM memory space. We needed a new approach to partitioning the memory to include partitioning the VMMs memory space.

Through discussions with UI, we came up with the following solution; modify the page tables created by the VMM boot code. Each core sets up a set of page tables for paging. These page tables are first created in real mode with a direct mapping of all addresses in the 4 GB of physical memory and then the VMM jumps to these tables when switching to protected mode. Hereafter all addresses go through the page tables to access physical memory and all addresses are considered virtual. We needed to modify the page tables to only map pages that are associated with the specific partitions of physical memory for each core.

The original code used assembler macros to build the page tables. At first, we tried to use the .align directive of the assembler to force the page tables to align to partition sizes. However, .align directives inside macros do not work. In the end, we removed the macro that created the

[7] Increased from 32K partitions to 37K partitions to account for increase from 4GB DRAM to 4.768GB DRAM.

page directory. This macro prevented the target addresses from being moved or from the distance between two targets in the same macro from changing. We had to break up the macro and assign each page table to its own section using linker directives. Each section would then be arranged in the linker script file. We only created page tables for the pages that mapped to the partitions belonging to the particular core that was creating the page table. This caused problems later because when the bootloader places the VMM code in memory and the VMM creates its own page table mappings based on the partitions, the addresses are not one-to-one and would need to adjust each memory reference for new addresses. Instead, we redesigned the page tables to include all pages but labeled the pages that belong to other partitions as not present.

We now needed a way to place the VMM code at specific locations, place the page tables at specific locations, and have each core execute its own version of the VMM code. To accomplish this, we again employed the linker script to place the page tables within their respective partitioned memory space. We also reduced the code size by forcing all the processors to execute the same code, except for bootup where the page tables are built. In real-mode, the BootStrap Processor (BSP) would then copy the data and code segments of the VMM from its partitioned memory space to that of each of the other cores' partitioned memory spaces. Using the label mapping from Appendix B, we were able to see that all the code for the VMM resides within a single partition. Additionally, by copying the code, the memory offsets remain the same since the compiler uses relative addressing instead of direct addressing. The only times we needed to force offsets on the code was after each cores bootup. We forced the offset to jump to the start of the VMM code within its own partition, and to point to their guest software; which was placed after the VMM code. The linker script used is shown in Appendix C.

Interleaving the partitions allowed us to reduce the impact of code duplication caused by each core executing its own VMM code. Table 7 shows a comparison of the number of lines of code (LOC) and the size of the code for the Multi-core MILS hypervisor, a quad-core version of the IAVMM code, and the original IAVMM code.

Table 7: Comparison of Multi-core MILS hypervisor and IAVMM

	LOC (asm)	LOC (c)	Physical Size
Multi-core MILS Hypervisor	6293	5322	40.0 MB
Multi-core IAVMM	4909	5171	38.7 MB
Original IAVMM	454	4329	16.2 MB

Up to this point, we were able to boot each core with each core executing its own VMM code within its partitioned memory. Additionally, we were still partitioning the e820 memory map. We provided each core's bitmap with only its partitioned usable e820 memory locations. We set up the extended page table (EPT) to map guest addresses to the partitioned memory addresses used by the bitmap. This way the guest would see one continuous address space even though it really was working on a partitioned address space. Unfortunately, this caused page faults when launching the virtual machine. Again this was due to a mismatch in the different address translations.

When the guest addresses were converted to virtual host addresses by the EPT they mapped to pages that did not belong to the cores partitioned memory. Normally this would not be an issue because the guest software would reside on its own stack; however, the IAVMM hypervisor did not include separate guest services. The guest software was actually a VMM function call that would reference other VMM functions (similar to kernel functions) to perform operations such as displaying text to the monitor. If the VMM had booted the guest software within its own address space as commercial hypervisors do, then the EPT could map only the memory partitions that belong to the core. As this is not the case, we had to look at several possible solutions:

1. Modify the guest OS so that it executes within a single partition without calling to any VMM functions and recalculate the addresses based on the EPT mapping structure. This is not an optimal solution as it does not scale well for other applications.
2. Have the guest trap back to the VMM for all address resolutions. This option is also not optimal as it eliminates the benefits of the EPT.
3. Make the EPT structure map to all memory and label the pages that do not belong to a cores' partition as not present. This provided the most optimal solution. A side effect of this solution was removing the complexity of the e820 mapping structure and instead replacing it with a more complex EPT structure. Appendix D shows the updated e820 and EPT structures. This new method created a more simplistic approach to partitioning and made it possible to allow larger physical memory, where the bitmap could extend past a partition.

We were able to include additional security when designing the EPT structure by labeling the usable memory locations that had to be shared as not present. This allowed the VMM to use those locations if needed, such as video frame buffer writing, but prevents the guest software from accessing those locations. The VMM functions, such as printf, were duplicated so that each core would call its own "kernel" code which resides at different locations. We also had to disable all spin-lock mechanisms used for synchronizing the four cores upon bootup because synchronization requires shared memory.

The final version of the Multi-core MILS hypervisor boots all four cores of the i7-860 quad-core processor, with each core running its own VMM on its own partitioned memory. This technique allows for the Multi-core MILS hypervisor to partition the cache, eliminating the covert channel.

4.0 RESULTS AND DISCUSSION

This section presents ideas and results for testing the final versions of both the Multi-core MILS processor and the Multi-core MILS Hypervisor. Each of these products will be discussed in their respective subsections.

4.1 Task 1: Multi-core MILS Processor

The limited scope of the project had our development solely focus on the Xilinx EDK platform. The main rationale was due to the OpenSPARC processors having pre-built projects for the processors. However, no project was able to run out of the box with our EDK suites. Therefore, future efforts should attempt to either use the same version (ISE 10.1) that the OpenSPARC processors specify.

4.1.1 Future Research Reading List. The following section is a collection of interesting whitepapers and documents that we feel pertain to future development of the MMP. These documents were found near the end of the development cycle, but the potential impact of future development has been weighed as significant

1. "Architecture for Protecting Critical Secrets in Microprocessors." [22] This document talks about presenting new concepts in microprocessor security that may allow for secure mechanisms in on-chip cache and on-chip secrets. Although the emphasis in focused on "on-line" interactions and some higher level aspects of computing, outside of the cache, there are interesting discussions relating to security at a low level/hardware stance.
2. United States patents relating to secure computing. Several patents may exist that are close to, if not the same as, the methodology planned in the NCID integration for the MMP.
   a. 7,107,459 - Secure Memory Access System and Method [23]
   b. 7,496,727 - Secure CPU and Memory Management Unit with Cryptographic Extensions [24]
   c. 6,523,118 - Secure Cache for Instruction and Data Protection [25]
3. Previous efforts on the OpenSPARC T1 regarding security. In 2011 a new article discussed past efforts of prototyping secure processing on the OpenSPARC with the collaboration of individuals from the City University of Hong Kong and Princeton University. A major difference was in the addition of an encryption/decryption module leveraging the Advanced Encryption Standard (AES) 128-bit Cipher Block Chaining (CBC) in protecting data stemming from the MicroBlaze controller. This gives a tantalizing view of secure computing with little difference in the speed of execution leveraging an add-on module to the MicroBlaze soft-core [26].

4.1.2 Future Research Recommendations. Upon completing this report, no effort has been made to integrate the NCID via the Verilog source code. Cursory viewing allows us to recommend the following:

1. First, attempt to modify the C code executed on the MicroBlaze soft-core processor instead of modifying the MicroBlaze itself. Originally, we thought the cache crossbar (CCX) of the OpenSPARC T1 was implemented in Verilog. However, the MicroBlaze C code actually implements the CCX to control cache traffic. In a way, this makes things quite a bit easier in the sense that the MicroBlaze emulates the cache.

   In order to implement the NCID cache scheme on the MicroBlaze, there are a few files in the 'ccx-firmware' folder that need to be modified. They are: 'mbfw_main', 'mbfw_pcx_cpx' and 'mbfw_reverse_dir'. All three files are sections of C code that are used to emulate the CCX. The 'mbfw_pcx_cpx' and 'mbfw_reverse_dir' files look to be the most important of the three files, as they define how the cache operates.

   Although it is still somewhat hard to speculate further into what else within the OpenSPARC would need to be modified in order to get the NCID working, it is still feasible that no changes to the underlying HDL would be necessary. The underlying structure of the CCX, as far as inputs and outputs would still be the same; there would just be more rules pertaining to cache traffic.

2. Another option is to not use the OpenSPARC T1 processor. The rationale for choosing the OpenSPARC T1 was due to the processor having an integrated MicroBlaze soft-core processor using the C language. However, moving forward using a different processor might be a better alternative to the current OpenSPARC T1 decision. The OpenSPARC T2 is a viable candidate, and any other processor must be open source and have the ability to be experimented on using a hardware development platform (e.g. FPGA's).

   The OpenSPARC T2 seemingly is a better processor than the T1, considering the processor is completely developed on an FPGA and no temporary development modules are used in its implementation (ex. The MicroBlaze soft-core).

4.1.3 Future Test Recommendations. Applying the NCID architecture to MILS applications requires the guarantee that data of varying security classifications do not reside on the same shared cache resource. To test this, we can exploit the ability of the FPGA to "snoop" on internal signals that normally would not be accessible to the user. Using this feature, we can view the internal state of the cache hierarchy as different cores access different data sets.

To show that the NCID architecture functions correctly, we would have one or more cores reading a set of varying classified data. As the data is read, we would keep track of which data is stored in the LLC and which is stored in the lower level cache. The test should show all data that is of any classification only stored in the lower level cache while their respective tags are stored in the LLC. Additionally all data that is unclassified can be stored in the LLC as well as the lower level cache.

It is important to note that the NCID architecture is not designed to defend against covert channels such as the cache timing attack as described by Percival [11]. This architecture is only to guarantee that data from different classifications do not reside on the same shared resource.

4.2 Task 2: Multicore MILS Hypervisor

We developed and tested the Multi-core MILS Hypervisor using two development systems. The first system, a Dell Precision M6500 Mobile Workstation that uses an Intel Core2 i7 Quad-Core processor with 4 GB RAM, was used to develop the hypervisor code in GNU GCC version Ubuntu 4.4.3. The second system, a parts-built workstation, employed an Intel Core i7-860 Quad-Core processor with 8 MB LLC, 4 GB RAM, and the Intel Extreme DP55KG motherboard. This system was used solely for testing purposes. It would boot the grand unified bootloader (GRUB) from an external USB to load the VMM.

To test the final version of the Multi-core MILS hypervisor, we needed baseline comparison values. Our baseline comparison was a shared cache version of the IAVMM provided by UI. We modified the IAVMM provided by UI to invoke all four cores of the i7-860 quad-core processor. We also modified the output code to provide output display of all four cores on the single monitor. After verifying that this version of the IAVMM still functioned correctly, we then started looking into recording and testing methods.

One of the primary goals of the project was to provide a means to thwart cache timing attacks, an attack that takes advantage of the covert channel provided by the shared cache of multi-core processors. A cache timing attack requires two cores to share information via covert channels. The covert channel medium in this case is the LLC miss count/rate. Percival [11] details the inner-workings of the cache timing attack. Cache timing attacks are based on one core filling the cache and checking how long each read takes. The idea is that another core would intermittently evict the first core's data when it needed the cache, providing the first core with some information. An LLC miss takes longer time than the other cache misses. Using the partitioning technique developed for the Multi-core MILS Hypervisor, each core will have a dedicated portion of the cache and the data from one core should not be able to evict the data from another core in the cache. When the first core tries to access its data again, it will register an LLC miss. Intel provides internal performance monitoring counters accessible by software to keep track of the number of cache misses [18]. The architectural performance monitor, L3_LAT_CACHE.MISS (Event: 2E, Umask: 41), counts the number of LLC misses.

To prove that our Multi-core MILS hypervisor partitions the cache so that memory references from one core do not evict another core's data in the cache, we came up with the following tests.

Test 1: Have Core 0 read 8 MB of memory repeatedly, recording the number of LLC misses during each read[8]. This will provide a baseline number of cache misses for reading 8 MB for both the shared and partitioned versions of the hypervisor. It is expected that for the shared version this test will show roughly 1/4th the number of cache misses as the partitioned version. This is because the partitioned version will only have 2 MB of cache available per core, whereas the shared version will have all 8 MB available for a single core.

Test 2: Have Core 1 repeat the Test 1 procedures for both shared and partitioned versions. This test is more of a sanity check, showing that each core will act the same. The same results are expected.

Test 3: Have Core 1 read 2 MB of memory repeatedly, recording the number of LLC misses during each read. For both the shared version and the partitioned version the number of LLC misses should be roughly the same, with the partitioned version experiencing slightly higher misses due to code reads. The first read will report 2 MB of LLC misses and each subsequent read should report close to 0. These values will also provide a baseline for Tests 4, 5, and 6.

Test 4: Have both Core 0 and Core 1 repeatedly read 2 MB of memory each. After each read record the number of LLC misses. This test tries to detect when data from one core evicts the data belonging to another core. It is expected that the partitioned version results remain the same, per core, as Test 3. However, for the shared version it was expected to show an increase in the number of misses per core compared to Test 3 results.

Test 5: Repeat Test 4 but include all four cores. Same results as Test 4 are expected but with an increased number of LLC misses for the shared partition. Test 4 and 5 should show correct partitioning of cache memory.

[8] For all the tests it is expected that the first time the memory is read, the results will show a large number of cache misses due to the processor filling the cache for the first time. LLC misses occurred during the second iteration of reading the memory indicate the number of LLC evictions.

Test 6[9]: Force Core 0 to repeatedly read a large amount of memory, greater than 16 MB. Have Core 1 repeatedly read 2 MB of memory and report the number of LLC misses per read. This test simulates a cache timing attack. It is expected that the shared version should report 2 MB's worth of cache misses each time Core 1 reads 2 MB. The partitioned version should report the same results as Test 4.

When we first started testing, using the L3_LAT_CACHE.MISS performance counter, the performance counter was showing more LLC misses than memory operations were executed. Through further testing, we found out that the code to read the counters was actually invoking additional memory reads. We simplified the code to invoke only two extra memory operations per measured memory operation. The code to perform the memory reads for the partitioned version and the shared version are shown below. We increment per page because the prefetcher fetches the entire page when a cache miss occurs. Otherwise reading 1 page (4 KB) of memory using consecutive addresses would result in a single cache line miss despite the fact that there are 64 cache lines per page. Reading 512 addresses, each a page apart, results in causing 2 MB of data loaded into the cache. For the partitioned version, we increment by 0x60000 to jump to the next partition assigned to the core. By doubling the value stored in rdx for the partitioned version and rcx for the shared version, we can increase the amount of memory read.

```
//partitioned version memory read code
for (l = 0; l < NUM_READS; l++)
{
   rdmsr(IA32_PMC1, p1, p2);  //read performance counter
   __asm__ __volatile__
   (
      "mov $16, %%rdx \n"     //set outer loop to 16
      "mov $0x40020000, %%rax\n"    //set start address
      "ap1_label_2:\n"  //begin outer loop
      "mov $32, %%rcx \n"           //set inner loop variable
      "ap1_label_3:\n"  //begin inner loop
      "mov (%%rax), %%rbx\n"  //read pointer
      "add $4096, %%rax\n"    //increment to next page
      "loop ap1_label_3\n"    //inner loop
      "add $0x60000, %%rax\n"       //jump to next memory block
      "sub $1, %%rdx\n" //decrement outer loop variable
      "and %%rdx, %%rdx\n"    //check outer loop condition
      "jnz  ap1_label_2\n"    //outer loop if not zero
      : /*no output*/ : /*no input*/ : "%rcx", "%rax", "%rbx", "%rdx"
   );
   rdmsr(IA32_PMC1, c1, c2);
   previous[1][l] = p1 | (u64)p2<<32;
   current[1][l] = c1 | (u64)c2<<32;
}


//shared version memory read code
for (l = 0; l < NUM_READS; l++)
{
   rdmsr(IA32_PMC1, p1, p2);
```

[9] Test 6 cannot be implemented with uncore performance counters.

```
   __asm__ __volatile__
   (
      "mov $512, %%rcx \n"
      "mov $0x40000000, %%rax\n"
      "ap1_label_2:\n"
      "mov (%%rax), %%rbx\n"
      "add $4096, %%rax\n"
      "loop ap1_label_2"
      : /*no output*/ : /*no input*/ : "%rcx", "%rax", "%rbx"
   );
   rdmsr(IA32_PMC1, c1, c2);
   previous[1][l] = p1 | (u64)p2<<32;
   current[1][l] = c1 | (u64)c2<<32;
}
```

Tables 8-11 show the results from the L3_LAT_CACHE.MISS event tests. Full results are shown in Appendix E. Tables 8 & 9 show that Tests 1 and 2 did not provide the expected results. Since Tests 1 and 2 read 8 MB of information per loop it was expected that the partitioned version would show roughly 4 times the number of LLC misses than the shared version. However, the difference is only around 320 misses. The large number of misses for the first run of the loop indicates the cache filling for the first time and cannot be used for showing LLC misses caused by evictions.

Table 8: Test 1 Results

Run #	Shared Misses	Partitioned Misses	Difference
1	43030	43387	357
2	12340	12691	351
3	12332	12653	321
4	12332	12652	320
5	12332	12652	320
6	12332	12652	320
7	12332	12652	320
8	12332	12652	320
9	12332	12652	320
10	12332	12652	320
11	12332	12652	320
12	12332	12652	320
13	12332	12652	320
14	12332	12652	320
15	12332	12652	320
16	12332	12652	320
17	12332	12652	320
18	12332	12652	320
19	12332	12652	320
20	12332	12652	320

Table 9: Test 2 Results

Run #	Shared Misses	Partitioned Misses	Difference
1	43038	43387	349
2	12346	12701	355
3	12334	12660	326
4	12334	12654	320
5	12334	12654	320
6	12334	12654	320
7	12334	12654	320
8	12334	12654	320
9	12334	12654	320
10	12334	12654	320
11	12334	12654	320
12	12334	12654	320
13	12334	12654	320
14	12334	12654	320
15	12334	12654	320
16	12334	12654	320
17	12334	12654	320
18	12334	12654	320
19	12334	12654	320
20	12334	12654	320

Table 10 shows the results from Tests 3, 4, and 5. These tests showed expected results. Test 3 was to provide a baseline for both the shared and partitioned misses given a 2 MB read. The values in Test 3 would be subtracted from those of Test 4 and 5 to determine the number of increased misses when going from a single core to a dual core and a quad-core respectively. For Test 4 the increase is not very significant. When running both Core 0 and Core 1, each reading 2 MB, the shared version only noticed 3 additional misses while the partitioned only noticed 1. Although a larger increase in cache misses was expected for the shared version, Intel's SmartCache has the ability to dynamically allocate the cache so that it reduces the number of cache misses. What is more telling is the larger increase for Test 5. Here all four cores were reading 2 MB. With an 8 MB cache the cache should be filled. Also given that the cache contains code as well, part of the increase to the shared and partitioned LLC miss count is due to code misses and not only data misses. The number of cache misses for the shared cache is roughly double the number of misses than the partitioned cache. This indicates that some of the evictions on the shared cache are caused by data from other cores. Given these results, we took this test a little further and recorded the results of all four cores reading 4 MB and 8 MB, Test 7 and Test 8 respectively. The results are shown in Table 11. Figure 8 charts the differences in LLC miss increases between the shared version and the partitioned version as the number of cores increases and the amount of memory that is read is increased.

Table 10: Test 3, 4, and 5 Results

	Test 3		Test 4				Test 5			
Run #	Shared Misses	Partitioned Misses	Shared Misses	Partitioned Misses	Shared Increase	Partitioned Increase	Shared Misses	Partitioned Misses	Shared Increase	Partitioned Increase
1	10765	10855	10770	10855	5	0	12049	10987	1284	132
2	2866	3208	2871	3208.5	5	0.5	3127	3338.3	261	130.3
3	2106	3196	2109	3197	3	1	2363.8	3328.5	257.8	132.5
4	2106	3196	2109	3197	3	1	2363.8	3328.5	257.8	132.5
5	2106	3196	2109	3197	3	1	2363.8	3328.5	257.8	132.5
6	2106	3196	2109	3197	3	1	2363.8	3328.5	257.8	132.5
7	2106	3196	2109	3197	3	1	2363.8	3328.5	257.8	132.5
8	2106	3196	2109	3197	3	1	2363.8	3328.5	257.8	132.5
9	2106	3196	2109	3197	3	1	2363.8	3328.5	257.8	132.5
10	2106	3196	2109	3197	3	1	2363.8	3328.5	257.8	132.5
11	2106	3196	2109	3197	3	1	2363.8	3328.5	257.8	132.5
12	2106	3196	2109	3197	3	1	2363.8	3328.5	257.8	132.5
13	2106	3196	2109	3197	3	1	2363.8	3328.5	257.8	132.5
14	2106	3196	2109	3197	3	1	2363.8	3328.5	257.8	132.5
15	2106	3196	2109	3197	3	1	2363.8	3328.5	257.8	132.5
16	2106	3196	2109	3197	3	1	2363.8	3328.5	257.8	132.5
17	2106	3196	2109	3197	3	1	2363.8	3328.5	257.8	132.5
18	2106	3196	2109	3197	3	1	2363.8	3328.5	257.8	132.5
19	2106	3196	2109	3197	3	1	2363.8	3328.5	257.8	132.5
20	2106	3196	2109	3197	3	1	2363.8	3328.5	257.8	132.5

Table 11: Test Comparison for Tests 3, 7, and 8

	Test 3		Test 7				Test 8			
Run #	Shared Misses	Partitioned Misses	Shared Misses	Partitioned Misses	Shared Increase	Partitioned Increase	Shared Misses	Partitioned Misses	Shared Increase	Partitioned Increase
1	10765	10855	12042	10981.5	1277	126.5	12038.5	10978.8	1273.5	123.8
2	2866	3208	3354.5	3316.9	488.5	108.9	3342.3	3305.9	476.3	97.9
3	2106	3196	3349.9	3307.3	1243.9	111.25	3339	3295.5	1233	99.5
4	2106	3196	3349.9	3306.9	1243.9	110.9	3338.9	3295.4	1232.9	99.4
5	2106	3196	3349.9	3306.5	1243.9	110.5	3338.9	3295.3	1232.9	99.3
6	2106	3196	3349.9	3306.5	1243.9	110.5	3338.9	3295.3	1232.9	99.3
7	2106	3196	3349.9	3306.5	1243.9	110.5	3338.9	3295.3	1232.9	99.3
8	2106	3196	3349.9	3306.5	1243.9	110.5	3338.9	3295.3	1232.9	99.3
9	2106	3196	3349.9	3306.5	1243.9	110.5	3338.9	3295.3	1232.9	99.3
10	2106	3196	3349.9	3306.5	1243.9	110.5	3338.9	3295.3	1232.9	99.3
11	2106	3196	3349.9	3306.5	1243.9	110.5	3338.9	3295.3	1232.9	99.3
12	2106	3196	3349.9	3306.5	1243.9	110.5	3338.9	3295.3	1232.9	99.3
13	2106	3196	3349.9	3306.5	1243.9	110.5	3338.9	3295.3	1232.9	99.3
14	2106	3196	3349.9	3306.5	1243.9	110.5	3338.9	3295.3	1232.9	99.3
15	2106	3196	3349.9	3306.5	1243.9	110.5	3338.9	3295.3	1232.9	99.3
16	2106	3196	3349.9	3306.5	1243.9	110.5	3338.9	3295.3	1232.9	99.3
17	2106	3196	3349.9	3306.5	1243.9	110.5	3338.9	3295.3	1232.9	99.3
18	2106	3196	3349.9	3306.5	1243.9	110.5	3338.9	3295.3	1232.9	99.3
19	2106	3196	3349.9	3306.5	1243.9	110.5	3338.9	3295.3	1232.9	99.3
20	2106	3196	3349.9	3306.5	1243.9	110.5	3338.9	3295.3	1232.9	99.3

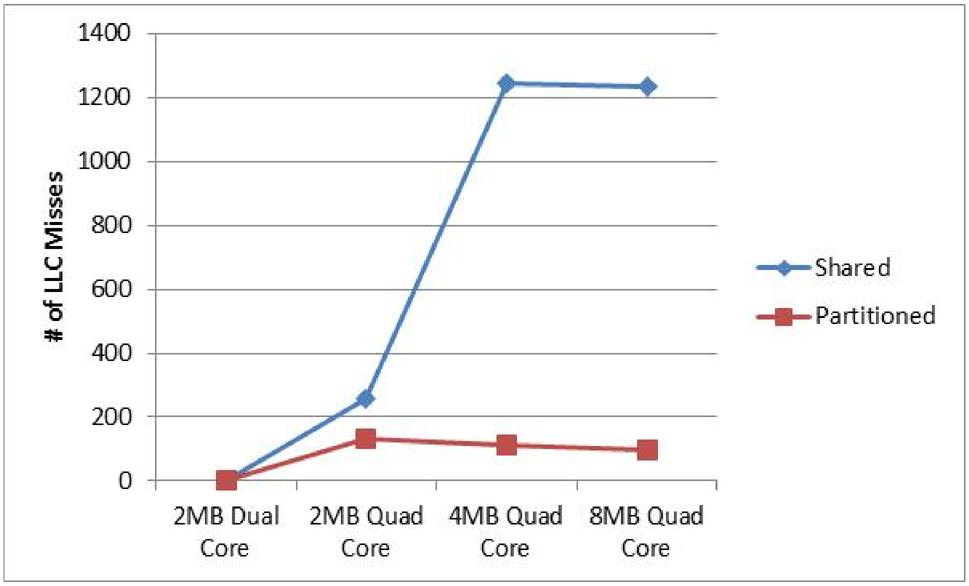

Figure 8: Shared vs. Partitioned LLC Miss Increase

As shown in Figure 8, the partitioned version, Multi-core MILS hypervisor, will actually show a smaller increase in LLC misses with larger data sets. This is caused by a cache separation among the cores. The data from one core will not evict the data of another. This does not mean that the partitioned version causes fewer misses. As shown in Appendix E, except for Test 7 the partitioned version always showed higher miss counts than the shared version. Again this is to be expected because by restricting the amount of cache available to each core, the instructions that reside in cache will also be evicted by fetched data, causing LLC misses for instructions. At the lower levels the difference between the shared and partitioned versions are negligible because of Intel's SmartCache.

Test 6 showed results similar to Test 4. Extending Test 6 to implement all four cores, Test 6b, did not provide the same increase as from Test 4 to Test 5. It is likely that Intel's SmartCache and Prefetch hardware have caused the test results to show little difference. There is no method to disable either feature on the Intel Core i7 processor. We suggest implementing the design on a processor that can disable these features, like the Intel Core Duo, in order to further test the Multi-core MILS hypervisor.

5.0 CONCLUSION

The goal of the project was to develop a means to reduce covert channels provided by multi-core architectures when implementing MILS applications. To this end, we have analyzed several cache hierarchy architectures and provided preliminary insight into the application of the NCID. Using a combination of OS data tagging and the NCID architecture we can develop a hardware implementation that will prevent sensitive information from being shared, even with an inclusive last level cache.

We have also developed a prototype Multi-core MILS hypervisor that executes on an Intel Core i7 processor. The hypervisor capitalizes on the fixed mapping between the physical RAM and the on-core cache. This mapping is based on the set associativity of the cache. Exploiting these features, we have partitioned the physical memory such that each core has access to a subset of the physical memory that will only map to a quarter of the cache.

The techniques developed in this project will allow the U.S. Air Force to process multiple classifications of information on a single processor/machine, effectively reducing the system resources required for missions. Additionally, the MILS system developed in this research will provide the increased security measures inherent in MILS.

6.0 RECOMMENDATIONS

As mentioned previously, the research towards the Multi-core MILS processor is incomplete. It is recommended to continue this research. We have presented additional readings that have become available since the project started that may provide enhanced approaches to solving the problem.

The Multi-core MILS hypervisor can be extended to allow for use on other processor families and models. Additionally the computations used to determine the partition sizes and bit locations can be performed in assembly. All the necessary information is provided by the CPUID. This will allow for the hypervisor to be ported to a larger set of machines. Further research can look into modifying the VMM design such that partitions can be dynamic. Such modifications would undoubtedly increase the code size. The hypervisor can be extended to boot a fully functional OS VM. This will simplify the generation of the EPT because the OS would manage its own memory space and the EPT can be viewed as a single continuous block of memory.

7.0 REFERENCES

[1] J. Rushby, "Design and Verification of Secure Systems," in *Proceedings of the 8th ACM Symposium on Operating System Principles*, 1981.

[2] C. Boettcher, K. Kung, J. Levowitz and K. Cariker, "The Benefits of Multi-Level Security," *Raytheon Technology Today,* vol. 2, pp. 8-11, 2007.

[3] C. Boettcher, R. DeLong, J. Rushby and W. Sifre, "The MILS Component Integration Approach to Secure Information Sharing," in *27th IEEE/AIAA Digital Avionics Systems Conference*, St. Paul, 2008.

[4] J. Alves-Foss, C. Taylor and P. Oman, "A Multi-Layered Approach to Security in High Assurance Systems," in *HICSS '04: Proceedings of the 37th Annual Hawaii International Conference on System Sciences*, 2004.

[5] Green Hills, "INTEGRITY Secure Virtualization," January 2003. [Online]. Available: http://www.ghs.com/products/rtos/integrity_virtualization.html. [Accessed January 2010].

[6] Wind River, "Wind River VxWorks MILS Platform 2.0," 2009. [Online]. Available: http://www.windriver.com/products/product-notes/PN_VE_MILS_Platform_0609.pdf. [Accessed January 2010].

[7] Lynux Works, "LynxSecure," 2010. [Online]. Available: http://www.lynuxworks.com/corporate/press/2010/lynxsecure-4.0.php. [Accessed January 2010].

[8] M. W. Vanfleet, J. A. Luke, W. R. Beckwith, C. Taylor, B. Calloni and G. Uchenick, "MILS: Architecture for High-Assurance Embedded Computing," *CrossTalk: Journal of Defense Software Engineering,* vol. 18, no. 8, pp. 12-16, 2005.

[9] T. Huffmire, B. Brotherton, N. Callegari, J. Valamehr, J. White, R. Kastner and T. Sherwood, "Designing secure systems on reconfigurable hardware," *ACM Trans. Des. Autom. Electron. Syst.,* vol. 13, no. 3, pp. 1-24, July 2008.

[10] D. Greer, "Industry Trends: Chip Makers Turn to Multicore Processors," *Computer,* pp. 11-13, May 2005.

[11] C. Percival, "Daemonic Dispatches," May 2005. [Online]. Available: http://www.daemonology.net/papers/htt.pdf. [Accessed August 2009].

[12] O. S. Saydjari, "On the Horizon: Multilevel Security: Reprise," *IEEE Security & Privacy,* 2004.

[13] R. Smith, "Introduction to Multilevel Security," December 2007. [Online]. Available: http://www.cs.stthomas.edu/faculty/resmith/r/mls/m3networks.html. [Accessed May 2009].

[14] L. Zhao, R. Iyer, S. Makineni, D. Newell and L. Cheng, "NCID: A Non-inclusive Cache, Inclusive Directory Architecture for Flexible and Efficient Cache Hierarchies," in *Proceedings of the 7th ACM International Conference on Computing Frontiers*, Bertinoro, 2010.

[15] Sun Microsystems Inc., *OpenSPARC™ T1 Processor Design and Verification Guide,* Santa Clara, California: Sun Microsystems, Inc, 2008.

[16] Sun Microsystems, Inc., "OpenSPARC T1 Microarchitecture Specification," August 2006. [Online]. Available: http://www.opensparc.net/opensparc-t1/index.html. [Accessed November 2011].

[17] R. A. Fisher and F. Yates, Statistical tables for biological, agricultural and medical research, Oliver and Boyd, Edinburgh, 1949.

[18] Intel Corporation, Intel 64 and IA-32 Architectures Software Developer's Manual, Santa Clara, 2010.

[19] K. Kaneda, "Tiny Virtual Machine Monitor," 2006. [Online]. Available: http://web.yl.is.s.utokyo.ac.jp/~kaneda/tvmm. [Accessed 27 March 2011].

[20] ACPI, "Advanced Configuration and Power Interface Specification, Revision 4.0a," 5 April 2010. [Online]. Available: http://www.acpi.info.

[21] Y. K. Okuji, B. Ford, E. S. Boleyn and K. Ishiguro, The Muliboot Specification version 0.6.96, Free Software Foundation, 1996.

[22] Princeton University, Department of Electrical Engineering, "Architecture for Protecting Critical Secrets in Microprocessors | PALMS - Princeton University," 2005. [Online]. Available: http://palms.ee.princeton.edu/PALMSopen/lee05architecture_w_cit.pdf. [Accessed 01 December 2011].

[23] Ludloff, C. D. Christian (San Jose, C. M. Kurt (San Jose and C. Andrew (Los Gatos, "Secure memory access system and method". Santa Clara, CA Patent 7,496,727, 24 Februrary 2009.

[24] Caronni, C. S. Germano (Palo Alto and C. Glenn (Mountain View, "Secure CPU and memory management unit with cryptographic extensions". Santa Clara, CA Patent 7,107,459, 12 September 2006.

[25] Buer and A. Mark Leonard (Gilbert, "Secure cache for instruction and data protection". Eindhoven, NL Patent 6,523,118, 18 Februrary 2003.

[26] Szefer and W. Z. ,. Y.-Y. C. ,. D. C. K. C. W. X. L. R. C. C. R. B. L. Jakub M., "Rapid_Single_Chip_Secure_Processor.pdf," in *2011 22nd IEEE International Symposium on Rapid System Prototyping (RSP)*, 2011.

[27] Sun Microsystems, Inc., "OpenSPARC T1 FPGA implementation Release 1.6 Update," [Online]. Available: http://www.opensparc.net/pubs/preszo/08/OpenSPARC_FPGA_Tutorial_16.pdf. [Accessed November 2011].

APPENDIX A: FIRST E820 MEMORY MAP PARTITIONING

The following is code used to partition the physical memory into 128KB partitions so as to partition the cache.

```
#include "types.h"
#include "failure.h"
#include "serial.h"
#include "page.h"
#include "multiboot.h"
#include "e820.h"
#include "string.h"

#define DEBUG 0
#define DEFAULT_VMM_PMEM_START (1 << 17) /*1MB = 0x100000 - Start VMM1 on Core 1 at 1MB */
#define DEFAULT_VMM_PMEM_END 0xccea6001
#define DEFAULT_VMM_PMEM_START2 (1 << 32) /*1GB = 0x100000000*/
#define DEFAULT_VMM_PMEM_END2 (13 << 28)+1

static void __init add_memory_region (struct e820_map *e820, u64 start, u64 size, enum e820_type type)
{
   if ( e820->nr_map >= E820_MAX_ENTRIES ) {
      fatal_failure ("Too many entries in the memory map.\n");
      return;
   }
   struct e820_entry *p = &e820->map [ e820->nr_map ];
   p->addr = start;
   p->size = size;
   p->type = type;
   e820->nr_map++;
}

void __init setup_memory_region (int cpuid, struct e820_map *e820, const struct multiboot_info
   *multi_boot_info)
{
   //first check that multi_boot_info->mmap_* fields are valid
   //check http://www.gnu.org/software/grub/manual/multiboot/html_node/Boot-information-format.html
   if (!(multi_boot_info->flags & multi_boot_info_MEMMAP))
      fatal_failure ("Bootloader provided no memory information.\n");
```

```
e820->nr_map = 0;
u64 temp = 0;
u64 index = 0;
int inner_done = 0;
u64 new_mmap_buf_addr = 0;
u64 new_mmap_buf_length = 0;
int counter = 0;

// go through memory_map entries in multi_boot_info, from mmap_addr to mmap_addr + mmap_length
while (index < multi_boot_info->mmap_length)
{
   const struct memory_map *mmap = (struct memory_map *) (multi_boot_info->mmap_addr + index);

   u64 mmap_buf_addr = ((u64)(mmap->base_addr_high) << 32) | (u64) mmap->base_addr_low;
   u64 mmap_buf_length = ((u64)(mmap->length_high) << 32) | (u64) mmap->length_low;

   if (mmap->type == E820_RAM) //only do calculations if usable memory region
   {
      inner_done = 0;
      do
      {
         temp = (mmap_buf_addr & MEM_SEPARATION_BITS)>>17; //separate bits 17 and 18
         if(((mmap_buf_addr < DEFAULT_VMM_PMEM_START) &&
               ((mmap_buf_addr + mmap_buf_length)  < DEFAULT_VMM_PMEM_START)) ||
                  (mmap_buf_addr >= DEFAULT_VMM_PMEM_END))
         {/*if start address is before start of vmm and end address is before the start of the vmm or
            the address is after the end of the vmm physical memory space save as reserved*/
            add_memory_region (e820, mmap_buf_addr, mmap_buf_length, E820_RESERVED);
            inner_done = 1;
         }
         else if ((mmap_buf_addr < DEFAULT_VMM_PMEM_START) &&
            ((mmap_buf_addr + mmap_buf_length) > DEFAULT_VMM_PMEM_START))
         {/*if start address is before start of vmm but end address is after start of vmm memory space
            then save first part as reserved and set new value for start and length*/
            //add memory region as E820_RESERVED type
            add_memory_region(e820, mmap_buf_addr, DEFAULT_VMM_PMEM_START-mmap_buf_addr, E820_RESERVED);
            mmap_buf_length -= (DEFAULT_VMM_PMEM_START-mmap_buf_addr);
            mmap_buf_addr = DEFAULT_VMM_PMEM_START;
         }
         else if ((u32)(cpuid >> 1) == (u32)(temp & 0x3))
```

```
{/*compare bits 17 and 18 of address to bits 1 and 2 of the cpuid,
   if the same then region corresponds to current cpuid and is usable.*/
   temp = mmap_buf_addr & MEM_SEPARATION_MASK; //check if lower 17 bits are zero
   if (temp > 0)
   {/*if the lower bits are not zero then region is not on a boundary address and length
      needs to be modified to show the length to the next region boundary address*/
      temp = mmap_buf_addr & MEM_SEPARATION_OFFSET;
      temp += MEM_SEPARATION_SIZE;
      new_mmap_buf_length = temp - mmap_buf_addr;
   }
   else
      new_mmap_buf_length = MEM_SEPARATION_SIZE;
   /*new_mmap_buf_length should now indicate how many bytes reside between the address and the
     next boundary address if new_mmap_buf_length indicates a size larger than read from the
     multiboot information structure then use the multiboots length and indicate that we have
     exhausted this region.*/
   if (new_mmap_buf_length > mmap_buf_length)
   {
      new_mmap_buf_length = mmap_buf_length;
      inner_done = 1;
   }
   new_mmap_buf_addr = mmap_buf_addr;

   //add memory region as E820_RAM type
   add_memory_region(e820, new_mmap_buf_addr, new_mmap_buf_length, mmap->type);

   counter++;
   if (inner_done == 0)
   {
      //go to next region of memory that corresponds to this cpuid
      mmap_buf_addr = (mmap_buf_addr & MEM_SEPARATION_OFFSET) + MEM_SEPARATION_NEXT;
      if (mmap_buf_length <= MEM_SEPARATION_NEXT)
      {//if there is not enough memory within this region to go to the next region
         add_memory_region(e820, new_mmap_buf_addr+new_mmap_buf_length, mmap_buf_length-
            new_mmap_buf_length, E820_RESERVED2);
         inner_done = 1; //indicate that all memory has been exhausted
      }
      else
      {// save range till next corresponding address range as reserved2
         mmap_buf_length -= MEM_SEPARATION_NEXT;
         add_memory_region(e820, new_mmap_buf_addr+new_mmap_buf_length, (MEM_SEPARATION_NEXT-
            MEM_SEPARATION_SIZE), E820_RESERVED2);
      }
   }
}
/*if the bits 17 and 18 of the address does not equal bits 1 and 2 of cpuid then the address
  does not correspond to the cpuid, need to go to next boundary region and check again.*/
else
{
   //if there is not enough space to go to next region
   if (mmap_buf_length <= MEM_SEPARATION_SIZE)
   {
      if(((mmap_buf_addr < DEFAULT_VMM_PMEM_START) &&
         ((mmap_buf_addr + mmap_buf_length) < DEFAULT_VMM_PMEM_START)) ||
            (mmap_buf_addr >= DEFAULT_VMM_PMEM_END))
         add_memory_region(e820, mmap_buf_addr, mmap_buf_length, E820_RESERVED);
      else
         add_memory_region(e820, mmap_buf_addr, mmap_buf_length, E820_RESERVED2);
      inner_done = 1; //indicate that all memory has been exhausted
   }
   else
   {
      temp = mmap_buf_addr & MEM_SEPARATION_MASK; //check if lower 17 bits are zero
      if (temp > 0)
      {/*if the lower bits are not zero then region is not on a boundary address and length
         needs to be modified to show the length to the next region boundary address*/
         temp = mmap_buf_addr & MEM_SEPARATION_OFFSET;
         temp += MEM_SEPARATION_SIZE;
         new_mmap_buf_length = temp - mmap_buf_addr;
      }
      else
         new_mmap_buf_length = MEM_SEPARATION_SIZE;
      /*new_mmap_buf_length should now indicate how many bytes reside between the address and
        the next boundary address*/
      if(((mmap_buf_addr < DEFAULT_VMM_PMEM_START) &&
         ((mmap_buf_addr + new_mmap_buf_length) < DEFAULT_VMM_PMEM_START)) ||
            (mmap_buf_addr >= DEFAULT_VMM_PMEM_END))
         add_memory_region(e820, mmap_buf_addr, new_mmap_buf_length, E820_RESERVED);
      else
         add_memory_region(e820, mmap_buf_addr, new_mmap_buf_length, E820_RESERVED2);
```

```
                  mmap_buf_length -= MEM_SEPARATION_SIZE;
               }
               mmap_buf_addr = (mmap_buf_addr & MEM_SEPARATION_OFFSET) + MEM_SEPARATION_SIZE;
            }
         }while(inner_done == 0);
      }
      else  //for each memory_map entry, add one corresponding entry to e820
         add_memory_region ( e820, mmap_buf_addr, mmap_buf_length, mmap->type );
      //go to next memory_map entry
      index += mmap->size + sizeof (mmap->size);
   }
}
```

APPENDIX B: LABEL MAPPINGS

```
0000000000100000 T _code
0000000000100000 T _start
0000000000100000 A phys
0000000000100000 T start
0000000000100008 t multiboot_header
0000000000100014 t die
0000000000100016 T physical_entry
00000000001002ce t virtual_entry
0000000000100308 T boot_gdt_descr
0000000000100312 t gdt_descr
0000000000101000 T gdt_table
0000000000101060 t multiboot_ptr
0000000000101064 T temporary_bsptest
0000000000103000 T vmx_asm_vmexit_handler
0000000000103008 t outb_p
0000000000103022 t cpuid_apic
0000000000103070 t cpuid_ebx
0000000000103089 t get_apicid
000000000010309c t setup_memory_init
000000000010319e T setup_memory
000000000010322f T print_pmem_layout
000000000010323c t displayProcessorInfo
00000000001032da t vms_create
0000000000103395 T bootAPs
0000000000103416 T start_vmm
000000000010351c T strstr
0000000000103573 T strlen
00000000001035a2 T strcmp
00000000001035f5 T strcpy
0000000000103636 T strncmp
000000000010369c T strchr
00000000001036d4 T strncpy
000000000010372d T strrchr
000000000010376f T strncasecmp
000000000010383a t __inline_memcpy
0000000000103890 t __bsp_inline_memcpy
00000000001038e6 t __ap1_inline_memcpy
000000000010393c t __ap2_inline_memcpy
0000000000103992 t __ap3_inline_memcpy
00000000001039e8 T memcpy
0000000000103a15 T memmove
0000000000103a94 T bsp_memmove
0000000000103b13 T ap1_memmove
0000000000103b92 T ap2_memmove
0000000000103c11 T ap3_memmove
0000000000103c90 T memset
0000000000103cd0 T bsp_memset
0000000000103d10 T ap1_memset
0000000000103d50 T ap2_memset
0000000000103d90 T ap3_memset
0000000000103dd0 t cpuid_ebx
0000000000103de9 t get_apicid
0000000000103dfc T clear_screen
0000000000103f82 t scroll
00000000001041b0 t bsp_scroll
0000000000104232 t ap1_scroll
00000000001042bc t ap2_scroll
000000000010433e t ap3_scroll
00000000001043c8 T putstr
00000000001045f6 T bsp_putstr
00000000001046bf T ap1_putstr
000000000010478e T ap2_putstr
0000000000104857 T ap3_putstr
0000000000104926 t number
0000000000104a30 t bsp_number
0000000000104b0f t ap1_number
0000000000104bee t ap2_number
0000000000104ccd t ap3_number
0000000000104dac T vsnprintf
0000000000105043 T bsp_vsnprintf
00000000001052da T ap1_vsnprintf
0000000000105571 T ap2_vsnprintf
0000000000105808 T ap3_vsnprintf
0000000000105a9f T strnlen
0000000000105aec t cpuid_ebx
0000000000105b05 t get_apicid
0000000000105b18 T breakpoint
0000000000105b49 t die
0000000000105b51 T fatal_failure
0000000000105b94 t cpuid_ebx
0000000000105bad t get_apicid
0000000000105bc0 t add_memory_region
0000000000105c50 T get_address
000000000010624e T e820_print_map
0000000000106428 T setup_memory_region
000000000010654b T get_nr_pages
0000000000106727 T get_max_pfn
0000000000106780 t cpuid_ebx
0000000000106799 t get_apicid
00000000001067ac t is_loadable_phdr
00000000001067db T get_elf_phdr
0000000000106811 T load_elf_image
000000000010692c t set_bit
0000000000106943 t cpuid
0000000000106983 t cpuid_eax
0000000000106999 t cpuid_ecx
00000000001069b2 t cpuid_edx
00000000001069cb t apic_mem_write
00000000001069e6 t apic_write
0000000000106a08 t apic_read
0000000000106a1d t apic_icr_write
0000000000106a4e t read_processor_info
0000000000106aa7 t check_vmx
0000000000106aec t display_cacheinfo
0000000000106cd7 t get_model_name
0000000000106d93 t get_cpu_vendor
0000000000106e00 t early_identify_cpu
0000000000107064 t init_intel
00000000001070e5 T enable_intel_vmx
000000000010713d T wakeup_secondary_core
0000000000107358 t cpuid
0000000000107398 t cpuid_apic
00000000001073e6 t cpuid_eax
00000000001073fc t fls
000000000010741f t get_count_order
0000000000107451 t cpuid4_cache_lookup
0000000000107506 t find_num_cache_leaves
000000000010755d T init_intel_cacheinfo
```

```
0000000000107a24 t apic_read
0000000000107a39 T get_maxlvt
0000000000107a7f T udelay
0000000000107ac0 t get_alloc_bitmap_idx
0000000000107ad2 t get_alloc_bitmap_offset
0000000000107ae3 t allocated_in_map
0000000000107b75 t map_alloc
0000000000107d9e t map_free
0000000000107fb4 t free_alloc_bitmap_region
000000000010803c T free_host_alloc_bitmap_region
000000000010806b t init_host_alloc_bitmap
000000000010822f T naive_memmap_init
000000000010847e t is_free_contiguous_region
00000000001084d7 t alloc_pages
0000000000108584 T alloc_host_pages
00000000001085c0 t cpuid_ebx
00000000001085d9 t get_apicid
00000000001085ec T pg_table_alloc
0000000000108637 t get_index_pae4kb
00000000001086d3 t get_4kb_entry_intel
0000000000108709 t entry4kb_is_existed
000000000010872e T __mmap_4kb_intel
00000000001089dc T mmap_pml4
0000000000108a24 t cpuid_ebx
0000000000108a3d t get_apicid
0000000000108a50 t rdtscl
0000000000108a6f t read_cr0
0000000000108a80 t read_cr2
0000000000108a91 t read_cr3
0000000000108aa2 t read_cr4
0000000000108ab3 t read_cr8
0000000000108ac5 t write_cr0
0000000000108ad6 t set_in_cr4
0000000000108aee t clear_in_cr4
0000000000108b0f t __vmptrld
0000000000108b24 t __vmptrst
0000000000108b35 t __vmclear
0000000000108b4b t __vmread
0000000000108b6b t __vmwrite
0000000000108b8b t __vmlaunch
0000000000108b94 t __vmresume
0000000000108bc7 t __vmxoff
0000000000108bd0 t __vmxon
0000000000108bf2 t init_vmx_ctrl
0000000000108c22 t init_vmcs_config
0000000000108cc4 t start_vmx
0000000000108e41 t stop_vmx
0000000000108e90 t construct_vmcs
000000000010a18c t vmx_create_vmcs
000000000010a205 T launch_mini_guest
000000000010a273 t print_failed_vmentry_reason
000000000010a2c3 T vmx_vmexit_handler
000000000010a76e T handle_init_sipi_sipi
000000000010a848 T alloc_vmcs
000000000010a893 T create_4kb_extended_pagetable
000000000010adea T vmx_create
000000000010ae50 T outf_bspap
000000000010af36 T outf
000000000010b01a T outf_ap1
000000000010b0fe T outf_ap2
000000000010b1e2 T outf_ap3
000000000010b2c8 t cpuid
000000000010b308 t read_cr0
000000000010b319 t write_cr0
000000000010b32a T idtCheck
000000000010b39e T gdtCheck
000000000010b412 T ldtCheck
000000000010b496 T strCheck
000000000010b4fe T testLDT
000000000010b571 T lslCheck
000000000010b5c7 T larCheck
000000000010b632 T verrCheck
000000000010b68d T popfCheck
000000000010b6b7 T pushfCheck
000000000010b705 T popCheck
000000000010b72e T pushCheck
000000000010b77a T movCR
000000000010b7a8 T taskswitchCheck
000000000010b7cc T retCheck
000000000010b7f0 T VMM2_Access_VMM1
000000000010b859 T VMM2_Access_VMM2
000000000010b8c2 T showsupported
000000000010b90d T instructiontest
000000000010b937 T storeHappen
000000000010b94d T mwaittest
000000000010b9ad T xgetbvtest
000000000010b9e7 T xsetbvtest
000000000010ba28 T getDRtest
000000000010bac4 T runCntLoop
000000000010bb0f T setCD
0000000000120000 T ap1_start
0000000000120000 T trampoline_ap1_start
00000000001200a0 t ap1_gdt_48
00000000001200a6 t ap1_gdt_descr
0000000000121000 T ap1_gdt_table
0000000000121060 T ap1_apicid_started
0000000000121060 t ap1_gdt_table_end
0000000000121068 T init_ap1_lock
0000000000121069 t ap1_promode
0000000000121069 T trampoline_ap1_end
000000000012107e t separate_line
0000000000121087 t sep
00000000001210d7 t print_msg2
00000000001210e8 t separate_line2
00000000001210ef t sep2
00000000001210f2 t print_msg4
00000000001210fc t msg4_loop
0000000000121108 t msg4_out
0000000000121115 t instruction_line
000000000012111e t instruction
0000000000121165 t print_msg3
0000000000121257 t AP1_callmain64
000000000012128f t ap1_multiboot_ptr
0000000000121293 T temporary_test
0000000000140000 T ap2_start
0000000000140000 T trampoline_ap2_start
00000000001400a0 t ap2_gdt_48
00000000001400a6 t ap2_gdt_descr
0000000000141000 T ap2_gdt_table
0000000000141060 T ap2_apicid_started
0000000000141060 t ap2_gdt_table_end
0000000000141068 T init_ap2_lock
0000000000141069 t ap2_promode
0000000000141069 T trampoline_ap2_end
```

```
000000000014115f t ap2_callmain64
0000000000141197 t ap2_multiboot_ptr
0000000000160000 T ap3_start
0000000000160000 T trampoline_ap3_start
00000000001600a0 t ap3_gdt_48
00000000001600a6 t ap3_gdt_descr
0000000000161000 T ap3_gdt_table
0000000000161060 T ap3_apicid_started
0000000000161060 t ap3_gdt_table_end
0000000000161068 T init_ap3_lock
0000000000161069 t ap3_promode
0000000000161069 T trampoline_ap3_end
000000000016115f t ap3_callmain64
0000000000161197 t ap3_multiboot_ptr
0000000000180000 t vmm_pml4_table
0000000000181000 t vmm_pdp_table
0000000000182000 t vmm_pdir_table
0000000000183000 t vmm_pdir_table1
0000000000184000 t vmm_pdir_table2
0000000000185000 t vmm_pdir_table3
0000000000186000 t vmm_pdir_table4
00000000001a0000 t vmm1_pml4_table
00000000001a1000 t vmm1_pdp_table
00000000001a2000 t vmm1_pdir_table
00000000001a3000 t vmm1_pdir_table1
00000000001a4000 t vmm1_pdir_table2
00000000001a5000 t vmm1_pdir_table3
00000000001a6000 t vmm1_pdir_table4
00000000001c0000 t vmm2_pml4_table
00000000001c1000 t vmm2_pdp_table
00000000001c2000 t vmm2_pdir_table
00000000001c3000 t vmm2_pdir_table1
00000000001c4000 t vmm2_pdir_table2
00000000001c5000 t vmm2_pdir_table3
00000000001c6000 t vmm2_pdir_table4
00000000001e0000 t vmm3_pml4_table
00000000001e1000 t vmm3_pdp_table
00000000001e2000 t vmm3_pdir_table
00000000001e3000 t vmm3_pdir_table1
00000000001e4000 t vmm3_pdir_table2
00000000001e5000 t vmm3_pdir_table3
00000000001e6000 t vmm3_pdir_table4
0000000000200000 t vmm_pde_table0_1
0000000000220000 t vmm1_pde_table0_1
0000000000240000 t vmm2_pde_table0_1
0000000000260000 t vmm3_pde_table0_1
0000000000280000 t vmm_pde_table0_2
00000000002a0000 t vmm1_pde_table0_2
00000000002c0000 t vmm2_pde_table0_2
00000000002e0000 t vmm3_pde_table0_2
0000000000300000 t vmm_pde_table0_3
0000000000320000 t vmm1_pde_table0_3
0000000000340000 t vmm2_pde_table0_3
0000000000360000 t vmm3_pde_table0_3
0000000000380000 t vmm_pde_table0_4
00000000003a0000 t vmm1_pde_table0_4
00000000003c0000 t vmm2_pde_table0_4
00000000003e0000 t vmm3_pde_table0_4
0000000000400000 t vmm_pde_table0_5
0000000000420000 t vmm1_pde_table0_5
0000000000440000 t vmm2_pde_table0_5
0000000000460000 t vmm3_pde_table0_5
```

```
0000000000480000 t vmm_pde_table0_6
00000000004a0000 t vmm1_pde_table0_6
00000000004c0000 t vmm2_pde_table0_6
00000000004e0000 t vmm3_pde_table0_6
0000000000500000 t vmm_pde_table0_7
0000000000520000 t vmm1_pde_table0_7
0000000000540000 t vmm2_pde_table0_7
0000000000560000 t vmm3_pde_table0_7
0000000000580000 t vmm_pde_table0_8
00000000005a0000 t vmm1_pde_table0_8
00000000005c0000 t vmm2_pde_table0_8
00000000005e0000 t vmm3_pde_table0_8
0000000000600000 t vmm_pde_table0_9
0000000000620000 t vmm1_pde_table0_9
0000000000640000 t vmm2_pde_table0_9
0000000000660000 t vmm3_pde_table0_9
0000000000680000 t vmm_pde_table0_10
00000000006a0000 t vmm1_pde_table0_10
00000000006c0000 t vmm2_pde_table0_10
00000000006e0000 t vmm3_pde_table0_10
0000000000700000 t vmm_pde_table0_11
0000000000720000 t vmm1_pde_table0_11
0000000000740000 t vmm2_pde_table0_11
0000000000760000 t vmm3_pde_table0_11
0000000000780000 t vmm_pde_table0_12
00000000007a0000 t vmm1_pde_table0_12
00000000007c0000 t vmm2_pde_table0_12
00000000007e0000 t vmm3_pde_table0_12
0000000000800000 t vmm_pde_table0_13
0000000000820000 t vmm1_pde_table0_13
0000000000840000 t vmm2_pde_table0_13
0000000000860000 t vmm3_pde_table0_13
0000000000880000 t vmm_pde_table0_14
00000000008a0000 t vmm1_pde_table0_14
00000000008c0000 t vmm2_pde_table0_14
00000000008e0000 t vmm3_pde_table0_14
0000000000900000 t vmm_pde_table0_15
0000000000920000 t vmm1_pde_table0_15
0000000000940000 t vmm2_pde_table0_15
0000000000960000 t vmm3_pde_table0_15
0000000000980000 t vmm_pde_table0_16
00000000009a0000 t vmm1_pde_table0_16
00000000009c0000 t vmm2_pde_table0_16
00000000009e0000 t vmm3_pde_table0_16
0000000000a00000 t vmm_pde_table1_1
0000000000a20000 t vmm1_pde_table1_1
0000000000a40000 t vmm2_pde_table1_1
0000000000a60000 t vmm3_pde_table1_1
0000000000a80000 t vmm_pde_table1_2
0000000000aa0000 t vmm1_pde_table1_2
0000000000ac0000 t vmm2_pde_table1_2
0000000000ae0000 t vmm3_pde_table1_2
0000000000b00000 t vmm_pde_table1_3
0000000000b20000 t vmm1_pde_table1_3
0000000000b40000 t vmm2_pde_table1_3
0000000000b60000 t vmm3_pde_table1_3
0000000000b80000 t vmm_pde_table1_4
0000000000ba0000 t vmm1_pde_table1_4
0000000000bc0000 t vmm2_pde_table1_4
0000000000be0000 t vmm3_pde_table1_4
0000000000c00000 t vmm_pde_table1_5
0000000000c20000 t vmm1_pde_table1_5
```

```
0000000000c40000 t vmm2_pde_table1_5
0000000000c60000 t vmm3_pde_table1_5
0000000000c80000 t vmm_pde_table1_6
0000000000ca0000 t vmm1_pde_table1_6
0000000000cc0000 t vmm2_pde_table1_6
0000000000ce0000 t vmm3_pde_table1_6
0000000000d00000 t vmm_pde_table1_7
0000000000d20000 t vmm1_pde_table1_7
0000000000d40000 t vmm2_pde_table1_7
0000000000d60000 t vmm3_pde_table1_7
0000000000d80000 t vmm_pde_table1_8
0000000000da0000 t vmm1_pde_table1_8
0000000000dc0000 t vmm2_pde_table1_8
0000000000de0000 t vmm3_pde_table1_8
0000000000e00000 t vmm_pde_table1_9
0000000000e20000 t vmm1_pde_table1_9
0000000000e40000 t vmm2_pde_table1_9
0000000000e60000 t vmm3_pde_table1_9
0000000000e80000 t vmm_pde_table1_10
0000000000ea0000 t vmm1_pde_table1_10
0000000000ec0000 t vmm2_pde_table1_10
0000000000ee0000 t vmm3_pde_table1_10
0000000000f00000 t vmm_pde_table1_11
0000000000f20000 t vmm1_pde_table1_11
0000000000f40000 t vmm2_pde_table1_11
0000000000f60000 t vmm3_pde_table1_11
0000000000f80000 t vmm_pde_table1_12
0000000000fa0000 t vmm1_pde_table1_12
0000000000fc0000 t vmm2_pde_table1_12
0000000000fe0000 t vmm3_pde_table1_12
0000000001000000 t vmm_pde_table1_13
0000000001020000 t vmm1_pde_table1_13
0000000001040000 t vmm2_pde_table1_13
0000000001060000 t vmm3_pde_table1_13
0000000001080000 t vmm_pde_table1_14
00000000010a0000 t vmm1_pde_table1_14
00000000010c0000 t vmm2_pde_table1_14
00000000010e0000 t vmm3_pde_table1_14
0000000001100000 t vmm_pde_table1_15
0000000001120000 t vmm1_pde_table1_15
0000000001140000 t vmm2_pde_table1_15
0000000001160000 t vmm3_pde_table1_15
0000000001180000 t vmm_pde_table1_16
00000000011a0000 t vmm1_pde_table1_16
00000000011c0000 t vmm2_pde_table1_16
00000000011e0000 t vmm3_pde_table1_16
0000000001200000 t vmm_pde_table2_1
0000000001220000 t vmm1_pde_table2_1
0000000001240000 t vmm2_pde_table2_1
0000000001260000 t vmm3_pde_table2_1
0000000001280000 t vmm_pde_table2_2
00000000012a0000 t vmm1_pde_table2_2
00000000012c0000 t vmm2_pde_table2_2
00000000012e0000 t vmm3_pde_table2_2
0000000001300000 t vmm_pde_table2_3
0000000001320000 t vmm1_pde_table2_3
0000000001340000 t vmm2_pde_table2_3
0000000001360000 t vmm3_pde_table2_3
0000000001380000 t vmm_pde_table2_4
00000000013a0000 t vmm1_pde_table2_4
00000000013c0000 t vmm2_pde_table2_4
00000000013e0000 t vmm3_pde_table2_4
```

```
0000000001400000 t vmm_pde_table2_5
0000000001420000 t vmm1_pde_table2_5
0000000001440000 t vmm2_pde_table2_5
0000000001460000 t vmm3_pde_table2_5
0000000001480000 t vmm_pde_table2_6
00000000014a0000 t vmm1_pde_table2_6
00000000014c0000 t vmm2_pde_table2_6
00000000014e0000 t vmm3_pde_table2_6
0000000001500000 t vmm_pde_table2_7
0000000001520000 t vmm1_pde_table2_7
0000000001540000 t vmm2_pde_table2_7
0000000001560000 t vmm3_pde_table2_7
0000000001580000 t vmm_pde_table2_8
00000000015a0000 t vmm1_pde_table2_8
00000000015c0000 t vmm2_pde_table2_8
00000000015e0000 t vmm3_pde_table2_8
0000000001600000 t vmm_pde_table2_9
0000000001620000 t vmm1_pde_table2_9
0000000001640000 t vmm2_pde_table2_9
0000000001660000 t vmm3_pde_table2_9
0000000001680000 t vmm_pde_table2_10
00000000016a0000 t vmm1_pde_table2_10
00000000016c0000 t vmm2_pde_table2_10
00000000016e0000 t vmm3_pde_table2_10
0000000001700000 t vmm_pde_table2_11
0000000001720000 t vmm1_pde_table2_11
0000000001740000 t vmm2_pde_table2_11
0000000001760000 t vmm3_pde_table2_11
0000000001780000 t vmm_pde_table2_12
00000000017a0000 t vmm1_pde_table2_12
00000000017c0000 t vmm2_pde_table2_12
00000000017e0000 t vmm3_pde_table2_12
0000000001800000 t vmm_pde_table2_13
0000000001820000 t vmm1_pde_table2_13
0000000001840000 t vmm2_pde_table2_13
0000000001860000 t vmm3_pde_table2_13
0000000001880000 t vmm_pde_table2_14
00000000018a0000 t vmm1_pde_table2_14
00000000018c0000 t vmm2_pde_table2_14
00000000018e0000 t vmm3_pde_table2_14
0000000001900000 t vmm_pde_table2_15
0000000001920000 t vmm1_pde_table2_15
0000000001940000 t vmm2_pde_table2_15
0000000001960000 t vmm3_pde_table2_15
0000000001980000 t vmm_pde_table2_16
00000000019a0000 t vmm1_pde_table2_16
00000000019c0000 t vmm2_pde_table2_16
00000000019e0000 t vmm3_pde_table2_16
0000000001a00000 t vmm_pde_table3_1
0000000001a20000 t vmm1_pde_table3_1
0000000001a40000 t vmm2_pde_table3_1
0000000001a60000 t vmm3_pde_table3_1
0000000001a80000 t vmm_pde_table3_2
0000000001aa0000 t vmm1_pde_table3_2
0000000001ac0000 t vmm2_pde_table3_2
0000000001ae0000 t vmm3_pde_table3_2
0000000001b00000 t vmm_pde_table3_3
0000000001b20000 t vmm1_pde_table3_3
0000000001b40000 t vmm2_pde_table3_3
0000000001b60000 t vmm3_pde_table3_3
0000000001b80000 t vmm_pde_table3_4
0000000001ba0000 t vmm1_pde_table3_4
```

```
0000000001bc0000 t vmm2_pde_table3_4
0000000001be0000 t vmm3_pde_table3_4
0000000001c00000 t vmm_pde_table3_5
0000000001c20000 t vmm1_pde_table3_5
0000000001c40000 t vmm2_pde_table3_5
0000000001c60000 t vmm3_pde_table3_5
0000000001c80000 t vmm_pde_table3_6
0000000001ca0000 t vmm1_pde_table3_6
0000000001cc0000 t vmm2_pde_table3_6
0000000001ce0000 t vmm3_pde_table3_6
0000000001d00000 t vmm_pde_table3_7
0000000001d20000 t vmm1_pde_table3_7
0000000001d40000 t vmm2_pde_table3_7
0000000001d60000 t vmm3_pde_table3_7
0000000001d80000 t vmm_pde_table3_8
0000000001da0000 t vmm1_pde_table3_8
0000000001dc0000 t vmm2_pde_table3_8
0000000001de0000 t vmm3_pde_table3_8
0000000001e00000 t vmm_pde_table3_9
0000000001e20000 t vmm1_pde_table3_9
0000000001e40000 t vmm2_pde_table3_9
0000000001e60000 t vmm3_pde_table3_9
0000000001e80000 t vmm_pde_table3_10
0000000001ea0000 t vmm1_pde_table3_10
0000000001ec0000 t vmm2_pde_table3_10
0000000001ee0000 t vmm3_pde_table3_10
0000000001f00000 t vmm_pde_table3_11
0000000001f20000 t vmm1_pde_table3_11
0000000001f40000 t vmm2_pde_table3_11
0000000001f60000 t vmm3_pde_table3_11
0000000001f80000 t vmm_pde_table3_12
0000000001fa0000 t vmm1_pde_table3_12
0000000001fc0000 t vmm2_pde_table3_12
0000000001fe0000 t vmm3_pde_table3_12
0000000002000000 t vmm_pde_table3_13
0000000002020000 t vmm1_pde_table3_13
0000000002040000 t vmm2_pde_table3_13
0000000002060000 t vmm3_pde_table3_13
0000000002080000 t vmm_pde_table3_14
00000000020a0000 t vmm1_pde_table3_14
00000000020c0000 t vmm2_pde_table3_14
00000000020e0000 t vmm3_pde_table3_14
0000000002100000 t vmm_pde_table3_15
0000000002120000 t vmm1_pde_table3_15
0000000002140000 t vmm2_pde_table3_15
0000000002160000 t vmm3_pde_table3_15
0000000002180000 t vmm_pde_table3_16
00000000021a0000 t vmm1_pde_table3_16
00000000021c0000 t vmm2_pde_table3_16
00000000021e0000 t vmm3_pde_table3_16
0000000002200000 t vmm_pde_table4_1
0000000002220000 t vmm1_pde_table4_1
0000000002240000 t vmm2_pde_table4_1
0000000002260000 t vmm3_pde_table4_1
0000000002280000 t vmm_pde_table4_2
00000000022a0000 t vmm1_pde_table4_2
00000000022c0000 t vmm2_pde_table4_2
00000000022e0000 t vmm3_pde_table4_2
0000000002300000 t vmm_pde_table4_3
0000000002320000 t vmm1_pde_table4_3
0000000002340000 t vmm2_pde_table4_3
0000000002360000 t vmm3_pde_table4_3
0000000002380000 t vmm_pde_table4_4
00000000023a0000 t vmm1_pde_table4_4
00000000023c0000 t vmm2_pde_table4_4
00000000023e0000 t vmm3_pde_table4_4
0000000002400000 t vmm_pde_table4_5
0000000002420000 t vmm1_pde_table4_5
0000000002440000 t vmm2_pde_table4_5
0000000002460000 t vmm3_pde_table4_5
0000000002480000 t vmm_pde_table4_6
00000000024a0000 t vmm1_pde_table4_6
00000000024c0000 t vmm2_pde_table4_6
00000000024e0000 t vmm3_pde_table4_6
0000000002500000 t vmm_pde_table4_7
0000000002520000 t vmm1_pde_table4_7
0000000002540000 t vmm2_pde_table4_7
0000000002560000 t vmm3_pde_table4_7
0000000002580000 t vmm_pde_table4_8
00000000025a0000 t vmm1_pde_table4_8
00000000025c0000 t vmm2_pde_table4_8
00000000025e0000 t vmm3_pde_table4_8
0000000002600000 t vmm_pde_table4_9
0000000002620000 t vmm1_pde_table4_9
0000000002640000 t vmm2_pde_table4_9
0000000002660000 t vmm3_pde_table4_9
0000000002680000 t vmm_pde_table4_10
00000000026a0000 t vmm1_pde_table4_10
00000000026c0000 t vmm2_pde_table4_10
00000000026e0000 t vmm3_pde_table4_10
0000000002700000 t vmm_pde_table4_11
0000000002720000 t vmm1_pde_table4_11
0000000002740000 t vmm2_pde_table4_11
0000000002760000 t vmm3_pde_table4_11
0000000002780000 t vmm_pde_table4_12
00000000027a0000 t vmm1_pde_table4_12
00000000027c0000 t vmm2_pde_table4_12
00000000027e0000 t vmm3_pde_table4_12
000000000281f000 t start_stack
000000000283f000 t start_ap1_stack
000000000285f000 t start_ap2_stack
000000000287f000 t start_ap3_stack
0000000002880000 t rdtscl
000000000288001f T guestOSImage
0000000002880058 t bsp_label_1
000000000288005f t bsp_label_2
000000000288007b T guestOSImage4AP1
0000000002880153 t ap1_label_1
000000000288015a t ap1_label_2
00000000028802eb T guestOSImage4AP2
00000000028803c0 t ap2_label_1
00000000028803c7 t ap2_label_2
0000000002880557 T guestOSImage4AP3
000000000288062c t ap3_label_1
0000000002880633 t ap3_label_2
0000000002900000 R _rodata
00000000029000e0 R strcase_charmap
0000000002900200 R _ctype
00000000029004e0 r assocs
00000000029005c0 r assocs
00000000029005e0 r levels
00000000029005e4 r types
0000000002903900 D _data
0000000002903900 d cache_table
```

```
00000000029039e0 d cache_table
0000000002903aa8 D m
0000000002904000 A _edata
0000000002904058 d _GLOBAL_OFFSET_TABLE_
0000000002905000 B _bss
0000000002905000 b bsp_scr_info
0000000002905008 b ap1_scr_info
0000000002905010 b ap2_scr_info
0000000002905018 b ap3_scr_info
0000000002905020 b g_cpuid_ext_feat_info
0000000002905024 b num_cache_leaves
0000000002905028 b is_initialized.1434
0000000002905040 b host_naive_allocator
0000000002906000 b M_MSR_BITMAP
0000000002907000 b vmcs_rev_id
0000000002907004 b pin_based_vm_exec_ctrls
0000000002907008 b proc_based_vm_exec_ctrls
000000000290700c b proc_based_vm_exec_ctrls2
0000000002907010 b vm_exit_ctrls
0000000002907014 b vm_entry_ctrls
0000000002907020 b buf.1328
0000000002907420 b buf.1323
0000000002907820 b buf.1318
0000000002907c20 b buf.1313
0000000002908020 b buf.1308
0000000002908420 B pgt_entry_4kb_intel
0000000002908440 B __cacheline_aligned
0000000002980000 A _end
```

APPENDIX C: LINKER SCRIPT

```
/* check
   http://www.redhat.com/docs/manuals/ent
   erprise/RHEL-4-Manual/gnu-
   linker/simple-commands.html */
OUTPUT_FORMAT("elf64-x86-64", "elf64-x86-
   64", "elf64-x86-64")
OUTPUT_ARCH(i386:x86-64)
ENTRY(_start)

/* Program headers */
PHDRS
{
   text PT_LOAD; /* Indicate that this
   program header describes a segment to
   be loaded from the file */
}

phys = 0x100000;
SECTIONS
{
   .text : AT(phys)
   {
      _code = .;
      /* place bsp boot code before vmm
         setup code */
      boot.o(.text);
      setup.o(.text);
      string.o(.text);
      printf.o(.text);
      failure.o(.text);
      e820.o(.text);
      elf.o(.text);
      cpu.o(.text);
      cacheinfo.o(.text);
      apic.o(.text);
      alloc.o(.text);
      page.o(.text);
      vmcs.o(.text);
      vm.o(.text);
      serial.o(.text);
      testcase.o(.text);
      . = ALIGN(4096);
      /* align ap1, ap2, and ap3 boot
         code to their respective
         partitions */
      . = ALIGN(131072);
      boot.o(ap1_bootcode);
      . = ALIGN(262144);
      boot.o(ap2_bootcode);
      . = ALIGN(131072);
      boot.o(ap3_bootcode);
      /* align page tables to
   partitions. Each page table is 128KB
   in size. The page table’s 512 entries
   is comprised of 32 page partition
   blocks that set the page attributes
   for 32 pages per block. Partitioning
   allows for the interleaving of each
   core’s page tables. Each page
directory is made up of 16 page
tables. There are almost 4 complete
page directory pointers in the page
directory pointer table.*/
   . = ALIGN(524288);
   boot.o(bsp_pd0_pt0);
   . = ALIGN(131072);
   boot.o(ap1_pd0_pt0);
   . = ALIGN(262144);
   boot.o(ap2_pd0_pt0);
   . = ALIGN(393216);
   boot.o(ap3_pd0_pt0);
   . = ALIGN(524288);
   boot.o(bsp_pd0_pt1)
   boot.o(ap1_pd0_pt1)
   boot.o(ap2_pd0_pt1)
   boot.o(ap3_pd0_pt1)
   boot.o(bsp_pd0_pt2)
   boot.o(ap1_pd0_pt2)
   boot.o(ap2_pd0_pt2)
   boot.o(ap3_pd0_pt2)
   boot.o(bsp_pd0_pt3)
   boot.o(ap1_pd0_pt3)
   boot.o(ap2_pd0_pt3)
   boot.o(ap3_pd0_pt3)
   boot.o(bsp_pd0_pt4)
   boot.o(ap1_pd0_pt4)
   boot.o(ap2_pd0_pt4)
   boot.o(ap3_pd0_pt4)
   boot.o(bsp_pd0_pt5)
   boot.o(ap1_pd0_pt5)
   boot.o(ap2_pd0_pt5)
   boot.o(ap3_pd0_pt5)
   boot.o(bsp_pd0_pt6)
   boot.o(ap1_pd0_pt6)
   boot.o(ap2_pd0_pt6)
   boot.o(ap3_pd0_pt6)
   boot.o(bsp_pd0_pt7)
   boot.o(ap1_pd0_pt7)
   boot.o(ap2_pd0_pt7)
   boot.o(ap3_pd0_pt7)
   boot.o(bsp_pd0_pt8)
   boot.o(ap1_pd0_pt8)
   boot.o(ap2_pd0_pt8)
   boot.o(ap3_pd0_pt8)
   boot.o(bsp_pd0_pt9)
   boot.o(ap1_pd0_pt9)
   boot.o(ap2_pd0_pt9)
   boot.o(ap3_pd0_pt9)
   boot.o(bsp_pd0_pt10)
   boot.o(ap1_pd0_pt10)
   boot.o(ap2_pd0_pt10)
   boot.o(ap3_pd0_pt10)
   boot.o(bsp_pd0_pt11)
   boot.o(ap1_pd0_pt11)
   boot.o(ap2_pd0_pt11)
   boot.o(ap3_pd0_pt11)
   boot.o(bsp_pd0_pt12)
   boot.o(ap1_pd0_pt12)
boot.o(ap2_pd0_pt12)
boot.o(ap3_pd0_pt12)
boot.o(bsp_pd0_pt13)
boot.o(ap1_pd0_pt13)
boot.o(ap2_pd0_pt13)
boot.o(ap3_pd0_pt13)
boot.o(bsp_pd0_pt14)
boot.o(ap1_pd0_pt14)
boot.o(ap2_pd0_pt14)
boot.o(ap3_pd0_pt14)
boot.o(bsp_pd0_pt15)
boot.o(ap1_pd0_pt15)
boot.o(ap2_pd0_pt15)
boot.o(ap3_pd0_pt15)
boot.o(bsp_pd0_pt16)
boot.o(ap1_pd0_pt16)
boot.o(ap2_pd0_pt16)
boot.o(ap3_pd0_pt16)
boot.o(bsp_pd1_pt1)
boot.o(ap1_pd1_pt1)
boot.o(ap2_pd1_pt1)
boot.o(ap3_pd1_pt1)
boot.o(bsp_pd1_pt2)
boot.o(ap1_pd1_pt2)
boot.o(ap2_pd1_pt2)
boot.o(ap3_pd1_pt2)
boot.o(bsp_pd1_pt3)
boot.o(ap1_pd1_pt3)
boot.o(ap2_pd1_pt3)
boot.o(ap3_pd1_pt3)
boot.o(bsp_pd1_pt4)
boot.o(ap1_pd1_pt4)
boot.o(ap2_pd1_pt4)
boot.o(ap3_pd1_pt4)
boot.o(bsp_pd1_pt5)
boot.o(ap1_pd1_pt5)
boot.o(ap2_pd1_pt5)
boot.o(ap3_pd1_pt5)
boot.o(bsp_pd1_pt6)
boot.o(ap1_pd1_pt6)
boot.o(ap2_pd1_pt6)
boot.o(ap3_pd1_pt6)
boot.o(bsp_pd1_pt7)
boot.o(ap1_pd1_pt7)
boot.o(ap2_pd1_pt7)
boot.o(ap3_pd1_pt7)
boot.o(bsp_pd1_pt8)
boot.o(ap1_pd1_pt8)
boot.o(ap2_pd1_pt8)
boot.o(ap3_pd1_pt8)
boot.o(bsp_pd1_pt9)
boot.o(ap1_pd1_pt9)
boot.o(ap2_pd1_pt9)
boot.o(ap3_pd1_pt9)
boot.o(bsp_pd1_pt10)
boot.o(ap1_pd1_pt10)
boot.o(ap2_pd1_pt10)
boot.o(ap3_pd1_pt10)
boot.o(bsp_pd1_pt11)
boot.o(ap1_pd1_pt11)
boot.o(ap2_pd1_pt11)
boot.o(ap3_pd1_pt11)
boot.o(bsp_pd1_pt12)
boot.o(ap1_pd1_pt12)
boot.o(ap2_pd1_pt12)
boot.o(ap3_pd1_pt12)
boot.o(bsp_pd1_pt13)
boot.o(ap1_pd1_pt13)
boot.o(ap2_pd1_pt13)
boot.o(ap3_pd1_pt13)
boot.o(bsp_pd1_pt14)
boot.o(ap1_pd1_pt14)
boot.o(ap2_pd1_pt14)
boot.o(ap3_pd1_pt14)
boot.o(bsp_pd1_pt15)
boot.o(ap1_pd1_pt15)
boot.o(ap2_pd1_pt15)
boot.o(ap3_pd1_pt15)
boot.o(bsp_pd1_pt16)
boot.o(ap1_pd1_pt16)
boot.o(ap2_pd1_pt16)
boot.o(ap3_pd1_pt16)
boot.o(bsp_pd2_pt1)
boot.o(ap1_pd2_pt1)
boot.o(ap2_pd2_pt1)
boot.o(ap3_pd2_pt1)
boot.o(bsp_pd2_pt2)
boot.o(ap1_pd2_pt2)
boot.o(ap2_pd2_pt2)
boot.o(ap3_pd2_pt2)
boot.o(bsp_pd2_pt3)
boot.o(ap1_pd2_pt3)
boot.o(ap2_pd2_pt3)
boot.o(ap3_pd2_pt3)
boot.o(bsp_pd2_pt4)
boot.o(ap1_pd2_pt4)
boot.o(ap2_pd2_pt4)
boot.o(ap3_pd2_pt4)
boot.o(bsp_pd2_pt5)
boot.o(ap1_pd2_pt5)
boot.o(ap2_pd2_pt5)
boot.o(ap3_pd2_pt5)
boot.o(bsp_pd2_pt6)
boot.o(ap1_pd2_pt6)
boot.o(ap2_pd2_pt6)
boot.o(ap3_pd2_pt6)
boot.o(bsp_pd2_pt7)
boot.o(ap1_pd2_pt7)
boot.o(ap2_pd2_pt7)
boot.o(ap3_pd2_pt7)
boot.o(bsp_pd2_pt8)
boot.o(ap1_pd2_pt8)
boot.o(ap2_pd2_pt8)
boot.o(ap3_pd2_pt8)
boot.o(bsp_pd2_pt9)
boot.o(ap1_pd2_pt9)
boot.o(ap2_pd2_pt9)
boot.o(ap3_pd2_pt9)
boot.o(bsp_pd2_pt10)
boot.o(ap1_pd2_pt10)
boot.o(ap2_pd2_pt10)
boot.o(ap3_pd2_pt10)
boot.o(bsp_pd2_pt11)
boot.o(ap1_pd2_pt11)
boot.o(ap2_pd2_pt11)
boot.o(ap3_pd2_pt11)
boot.o(bsp_pd2_pt12)
boot.o(ap1_pd2_pt12)
boot.o(ap2_pd2_pt12)
boot.o(ap3_pd2_pt12)
boot.o(bsp_pd2_pt13)
boot.o(ap1_pd2_pt13)
boot.o(ap2_pd2_pt13)
boot.o(ap3_pd2_pt13)
boot.o(bsp_pd2_pt14)
boot.o(ap1_pd2_pt14)
boot.o(ap2_pd2_pt14)
boot.o(ap3_pd2_pt14)
boot.o(bsp_pd2_pt15)
boot.o(ap1_pd2_pt15)
boot.o(ap2_pd2_pt15)
boot.o(ap3_pd2_pt15)
boot.o(bsp_pd2_pt16)
boot.o(ap1_pd2_pt16)
boot.o(ap2_pd2_pt16)
boot.o(ap3_pd2_pt16)
boot.o(bsp_pd3_pt1)
boot.o(ap1_pd3_pt1)
boot.o(ap2_pd3_pt1)
boot.o(ap3_pd3_pt1)
boot.o(bsp_pd3_pt2)
boot.o(ap1_pd3_pt2)
boot.o(ap2_pd3_pt2)
boot.o(ap3_pd3_pt2)
boot.o(bsp_pd3_pt3)
boot.o(ap1_pd3_pt3)
boot.o(ap2_pd3_pt3)
boot.o(ap3_pd3_pt3)
boot.o(bsp_pd3_pt4)
boot.o(ap1_pd3_pt4)
boot.o(ap2_pd3_pt4)
boot.o(ap3_pd3_pt4)
boot.o(bsp_pd3_pt5)
boot.o(ap1_pd3_pt5)
boot.o(ap2_pd3_pt5)
boot.o(ap3_pd3_pt5)
boot.o(bsp_pd3_pt6)
boot.o(ap1_pd3_pt6)
boot.o(ap2_pd3_pt6)
boot.o(ap3_pd3_pt6)
boot.o(bsp_pd3_pt7)
boot.o(ap1_pd3_pt7)
boot.o(ap2_pd3_pt7)
boot.o(ap3_pd3_pt7)
boot.o(bsp_pd3_pt8)
boot.o(ap1_pd3_pt8)
boot.o(ap2_pd3_pt8)
boot.o(ap3_pd3_pt8)
boot.o(bsp_pd3_pt9)
boot.o(ap1_pd3_pt9)
boot.o(ap2_pd3_pt9)
boot.o(ap3_pd3_pt9)
boot.o(bsp_pd3_pt10)
boot.o(ap1_pd3_pt10)
boot.o(ap2_pd3_pt10)
boot.o(ap3_pd3_pt10)
```

```
boot.o(bsp_pd3_pt11)
boot.o(ap1_pd3_pt11)
boot.o(ap2_pd3_pt11)
boot.o(ap3_pd3_pt11)
boot.o(bsp_pd3_pt12)
boot.o(ap1_pd3_pt12)
boot.o(ap2_pd3_pt12)
boot.o(ap3_pd3_pt12)
boot.o(bsp_pd3_pt13)
boot.o(ap1_pd3_pt13)
boot.o(ap2_pd3_pt13)
boot.o(ap3_pd3_pt13)
boot.o(bsp_pd3_pt14)
boot.o(ap1_pd3_pt14)
boot.o(ap2_pd3_pt14)
boot.o(ap3_pd3_pt14)
boot.o(bsp_pd3_pt15)
boot.o(ap1_pd3_pt15)
boot.o(ap2_pd3_pt15)
boot.o(ap3_pd3_pt15)
boot.o(bsp_pd3_pt16)
boot.o(ap1_pd3_pt16)
boot.o(ap2_pd3_pt16)
boot.o(ap3_pd3_pt16)
boot.o(bsp_pd4_pt1)
boot.o(ap1_pd4_pt1)
boot.o(ap2_pd4_pt1)
boot.o(ap3_pd4_pt1)
boot.o(bsp_pd4_pt2)
boot.o(ap1_pd4_pt2)
boot.o(ap2_pd4_pt2)
boot.o(ap3_pd4_pt2)
boot.o(bsp_pd4_pt3)
boot.o(ap1_pd4_pt3)
boot.o(ap2_pd4_pt3)
boot.o(ap3_pd4_pt3)
boot.o(bsp_pd4_pt4)
boot.o(ap1_pd4_pt4)
boot.o(ap2_pd4_pt4)
boot.o(ap3_pd4_pt4)
boot.o(bsp_pd4_pt5)
boot.o(ap1_pd4_pt5)
boot.o(ap2_pd4_pt5)
boot.o(ap3_pd4_pt5)
boot.o(bsp_pd4_pt6)
boot.o(ap1_pd4_pt6)
boot.o(ap2_pd4_pt6)
boot.o(ap3_pd4_pt6)
boot.o(bsp_pd4_pt7)
boot.o(ap1_pd4_pt7)
boot.o(ap2_pd4_pt7)
boot.o(ap3_pd4_pt7)
boot.o(bsp_pd4_pt8)
boot.o(ap1_pd4_pt8)
boot.o(ap2_pd4_pt8)
boot.o(ap3_pd4_pt8)
boot.o(bsp_pd4_pt9)
boot.o(ap1_pd4_pt9)
boot.o(ap2_pd4_pt9)
boot.o(ap3_pd4_pt9)
boot.o(bsp_pd4_pt10)
boot.o(ap1_pd4_pt10)
```

```
    boot.o(ap2_pd4_pt10)
    boot.o(ap3_pd4_pt10)
    boot.o(bsp_pd4_pt11)
    boot.o(ap1_pd4_pt11)
    boot.o(ap2_pd4_pt11)
    boot.o(ap3_pd4_pt11)
    boot.o(bsp_pd4_pt12)
    boot.o(ap1_pd4_pt12)
    boot.o(ap2_pd4_pt12)
    boot.o(ap3_pd4_pt12)
    /* Each stack is one partition in
       size */
    . = ALIGN(524288);
    boot.o(bsp_stack)
    . = ALIGN(131072);
    boot.o(ap1_stack)
    . = ALIGN(262144);
    boot.o(ap2_stack);
    . = ALIGN(131072);
    boot.o(ap3_stack);
    . = ALIGN(524288);
    /* Place guest code after all VMM
       code */
    guestOS.o(.text);
    . = ALIGN(524288);
} : text /* Assign this section to the
`text` segment described by the
program header */
```

```
    .rodata : AT(phys + (_rodata - _code))
    {
        _rodata = .;
        *(.rodata)
       . = ALIGN(4096);
    } : text

    .data : AT(phys + (_data - _code))
    {
        _data = .;
        *(.data)
       . = ALIGN(4096);
        } : text
    _edata = .;

    .bss : AT(phys + (_bss - _code))
    {
        _bss = .;
        *(.bss.stack_aligned)
        *(.bss)
        *(COMMON)
        . = ALIGN(4096);
    } : text

    . = ALIGN(524288);/*multiboot
    information placed after this
    location*/
    _end = .;
}
```

APPENDIX D: REVISED E820 MAP & EPT STRUCTURE

```
void __init setup_memory_region ( struct e820_map *e820, const struct multiboot_info *multi_boot_info, int
   coreid )
{
   /* Includes the memory region from 0x100000000 to 0x130000000 */
   if (!(multi_boot_info->flags & MBI_MEMMAP))
   {
      outf_bspap (coreid, "Bootloader provided no memory information.\n");
      fatal_failure ("Bootloader provided no memory information.\n", coreid);
   }

   e820->nr_map = 0;

   u64 index = 0;

   /* go through memory_map entries in multi_boot_info, from mmap_addr to mmap_addr + mmap_length */
   while ( index < multi_boot_info->mmap_length)
   {
      const struct memory_map *mmap = (struct memory_map *) (multi_boot_info->mmap_addr + index);

      u64 mmap_buf_addr = ( (u64)(mmap->base_addr_high) << 32) | (u64) mmap->base_addr_low;
      u64 mmap_buf_length = ( (u64)(mmap->length_high) << 32) | (u64) mmap->length_low;
      // Mark ranges excluded from memory map as reserved
      if(mmap_buf_addr == 0x90000)
         mmap_buf_length += 0x40000;
      else if (mmap_buf_addr == 0xcf800000)
         mmap_buf_length += 0x28000000;
      add_memory_region(e820, mmap_buf_addr, mmap_buf_length, mmap->type);

      //go to next memory_map entry
      index += mmap->size + sizeof (mmap->size);
   }
}


unsigned long create_4kb_extended_pagetable (const int coreid, struct e820_map *e820)
{
   const u64 ept_ptr  = pg_table_alloc(coreid);
```

```
u64 current_page;
unsigned long i, k;

/* determine the page number where the VMM code starts for each core */
u64 vmm_first_page = ((coreid == BSP) ?
   PAGE_DOWN(get_address(DEFAULT_BSP_VMM_PMEM_START, DEFAULT_VMM_HEAP_OFFSET, e820, 1, coreid)) :
      ((coreid == AP1) ?
      PAGE_DOWN(get_address(DEFAULT_AP1_VMM_PMEM_START, DEFAULT_VMM_HEAP_OFFSET, e820, 1, coreid)) :
         ((coreid == AP2) ?
         PAGE_DOWN(get_address(DEFAULT_AP2_VMM_PMEM_START, DEFAULT_VMM_HEAP_OFFSET, e820, 1, coreid)) :
         PAGE_DOWN(get_address(DEFAULT_AP3_VMM_PMEM_START, DEFAULT_VMM_HEAP_OFFSET, e820, 1,
   coreid)))));
/* determine the page number where the VMM code ends for each core */
u64 vmm_last_page  = ((coreid == BSP) ?
   PAGE_UP(get_address(DEFAULT_BSP_VMM_PMEM_START, DEFAULT_VMM_PMEM_SIZE, e820, 0, coreid)) :
      ((coreid == AP1) ?
      PAGE_UP(get_address(DEFAULT_AP1_VMM_PMEM_START, DEFAULT_VMM_PMEM_SIZE, e820, 0, coreid)) :
         ((coreid == AP2) ?
         PAGE_UP(get_address(DEFAULT_AP2_VMM_PMEM_START, DEFAULT_VMM_PMEM_SIZE, e820, 0, coreid)) :
         PAGE_UP(get_address(DEFAULT_AP3_VMM_PMEM_START, DEFAULT_VMM_PMEM_SIZE, e820, 0, coreid)))));

/* Traverse the e820 memory map structure to provide an EPT mapping for all pages. */
for (i = 0; i < e820->nr_map; i++)
{
   const struct e820_entry *p = &e820->map[i];
   current_page = p->addr;
   if (p->type == E820_RAM)
   {
      if(((current_page < PMEM_START) && ((current_page + p->size)   < PMEM_START)) ||
         ((current_page >= PMEM_END) && (current_page < PMEM_START2) &&
         ((current_page + p->size) < PMEM_START2)) || (current_page >= PMEM_END2))
      {/* If the entire e820 entry is located outside of usable VM space then map all the pages
          pertaining to the entry as not present (0)*/
         for (k = 0; k < (p->size >> PAGE_SHIFT); k++)
            mmap_pml4 (coreid, ept_ptr, current_page + (k*PAGE_SIZE), current_page + (k*PAGE_SIZE), 0);
      }
      else
      {
         if ((current_page < PMEM_START) && ((current_page + p->size) > PMEM_START))
         {/* If part of the e820 entry is located outside of usable VM space but part of it is located
```

```
             within usable VM space then map outside space as not present and continue with inside
             space*/
            for (k = 0; k < ((PMEM_START - current_page) >> PAGE_SHIFT); k++)
               mmap_pml4 (coreid, ept_ptr, current_page+(k*PAGE_SIZE), current_page+(k*PAGE_SIZE), 0);
            current_page = PMEM_START;
         }
         else if ((current_page < PMEM_START2) && ((current_page + p->size) > PMEM_START2))
         {/* If part of the e820 entry is located outside of usable VM space but part of it is located
             within usable VM space then map outside space as not present and continue with inside
             space*/
            for (k = 0; k < ((PMEM_START2 - current_page) >> PAGE_SHIFT); k++)
               mmap_pml4 (coreid, ept_ptr, current_page+(k*PAGE_SIZE), current_page+(k*PAGE_SIZE), 0);
            current_page = PMEM_START2;
         }
         //I now have a e820 entry with a starting address inside usable VM space
         do
         {
            if (((current_page & MEM_SEP_BITS) >> 17) == (coreid >> 1))
            {/* If the address maps to core’s partition and the address does not map to VMM memory
                space, then map as present (1) otherwise as not present */
               if ((current_page > vmm_last_page) || ((current_page) < vmm_first_page))
                  mmap_pml4 (coreid, ept_ptr, current_page, current_page, 1);
               else
                  mmap_pml4 (coreid, ept_ptr, current_page, current_page, 0);
            }
            else
               mmap_pml4 (coreid, ept_ptr, current_page, current_page, 0);
            current_page += PAGE_SIZE;
         //continue till all pages in e820 entry have been mapped
         }while(current_page < (p->addr + p->size));
      }
   }
   else
   {// Map all non-RAM memory ranges
      for (k = 0; k < (p->size >> PAGE_SHIFT); k++)
         mmap_pml4 (coreid, ept_ptr, current_page + (k*PAGE_SIZE), current_page + (k*PAGE_SIZE), 1);
   }
}
//return pointer to new ept structure
return ept_ptr;
```

```
}
```

APPENDIX E: TEST RESULTS

Table 12: Test 1 LLC Miss Count Values

Test 1	L3_LAT_CACHE.MISS									
	Shared					Partitioned				
	Start Value	Stop Value	Difference	Delta	Sum	Start Value	Stop Value	Difference	Delta	Sum
	9	A81F	43030		43030	9	A984	43387		43387
	A844	D853	12303	37	12340	A9AB	DB17	12652	39	12691
	D876	1087F	12297	35	12332	DB3A	10C84	12618	35	12653
	108A2	138AB	12297	35	12332	10CA7	13DF0	12617	35	12652
	138CE	168D7	12297	35	12332	13E13	16F5C	12617	35	12652
	168FA	19903	12297	35	12332	16F7F	1A0C8	12617	35	12652
	19926	1C92F	12297	35	12332	1A0EB	1D234	12617	35	12652
	1C952	1F95B	12297	35	12332	1D257	203A0	12617	35	12652
	1F97E	22987	12297	35	12332	203C3	2350C	12617	35	12652
	229AA	259B3	12297	35	12332	2352F	26678	12617	35	12652
	259D6	289DF	12297	35	12332	2669B	297E4	12617	35	12652
	28A02	2BA0B	12297	35	12332	29807	2C950	12617	35	12652
	2BA2E	2EA37	12297	35	12332	2C973	2FABC	12617	35	12652
	2EA5A	31A63	12297	35	12332	2FADF	32C28	12617	35	12652
	31A86	34A8F	12297	35	12332	32C4B	35D94	12617	35	12652
	34AB2	37ABB	12297	35	12332	35DB7	38F00	12617	35	12652
	37ADE	3AAE7	12297	35	12332	38F23	3C06C	12617	35	12652
	3AB0A	3DB13	12297	35	12332	3C08F	3F1D8	12617	35	12652
	3DB36	40B3F	12297	35	12332	3F1FB	42344	12617	35	12652
	40B62	43B6B	12297	35	12332	42367	454B0	12617	35	12652

Table 13: Test 2 LLC Miss Count Values

Test 2	L3_LAT_CACHE.MISS									
	Shared					Partitioned				
	Start Value	Stop Value	Difference	Delta	Sum	Start Value	Stop Value	Difference	Delta	Sum
	B	A829	43038		43038	A	A985	43387		43387
	A854	D863	12303	43	12346	A9AC	DB22	12662	39	12701
	D888	10891	12297	37	12334	DB46	10C96	12624	36	12660
	108B6	138BF	12297	37	12334	10CBA	13E04	12618	36	12654
	138E4	168ED	12297	37	12334	13E28	16F72	12618	36	12654
	16912	1991B	12297	37	12334	16F96	1A0E0	12618	36	12654
	19940	1C949	12297	37	12334	1A104	1D24E	12618	36	12654
	1C96E	1F977	12297	37	12334	1D272	203BC	12618	36	12654
	1F99C	229A5	12297	37	12334	203E0	2352A	12618	36	12654

	229CA	259D3	12297	37	12334	2354E	26698	12618	36	12654
	259F8	28A01	12297	37	12334	266BC	29806	12618	36	12654
	28A26	2BA2F	12297	37	12334	2982A	2C974	12618	36	12654
	2BA54	2EA5D	12297	37	12334	2C998	2FAE2	12618	36	12654
	2EA82	31A8B	12297	37	12334	2FB06	32C50	12618	36	12654
	31AB0	34AB9	12297	37	12334	32C74	35DBE	12618	36	12654
	34ADE	37AE7	12297	37	12334	35DE2	38F2C	12618	36	12654
	37B0C	3AB15	12297	37	12334	38F50	3C09A	12618	36	12654
	3AB3A	3DB43	12297	37	12334	3C0BE	3F208	12618	36	12654
	3DB68	40B71	12297	37	12334	3F22C	42376	12618	36	12654
	40B96	43B9F	12297	37	12334	4239A	454E4	12618	36	12654

Table 14: Test 3 LLC Miss Count Values

	L3_LAT_CACHE.MISS									
	Shared					Partitioned				
	Start Value	Stop Value	Difference	Delta	Sum	Start Value	Stop Value	Difference	Delta	Sum
Test 3	9	2A16	10765		10765	9	2A70	10855		10855
	2A3B	3548	2829	37	2866	2A96	36F8	3170	38	3208
	356B	3D82	2071	35	2106	371B	4374	3161	35	3196
	3DA5	45BC	2071	35	2106	4397	4FF0	3161	35	3196
	45DF	4DF6	2071	35	2106	5013	5C6C	3161	35	3196
	4E19	5630	2071	35	2106	5C8F	68E8	3161	35	3196
	5653	5E6A	2071	35	2106	690B	7564	3161	35	3196
	5E8D	66A4	2071	35	2106	7587	81E0	3161	35	3196
	66C7	6EDE	2071	35	2106	8203	8E5C	3161	35	3196
	6F01	7718	2071	35	2106	8E7F	9AD8	3161	35	3196
	773B	7F52	2071	35	2106	9AFB	A754	3161	35	3196
	7F75	878C	2071	35	2106	A777	B3D0	3161	35	3196
	87AF	8FC6	2071	35	2106	B3F3	C04C	3161	35	3196
	8FE9	9800	2071	35	2106	C06F	CCC8	3161	35	3196
	9823	A03A	2071	35	2106	CCEB	D944	3161	35	3196
	A05D	A874	2071	35	2106	D967	E5C0	3161	35	3196
	A897	B0AE	2071	35	2106	E5E3	F23C	3161	35	3196
	B0D1	B8E8	2071	35	2106	F25F	FEB8	3161	35	3196
	B90B	C122	2071	35	2106	FEDB	10B34	3161	35	3196
	C145	C95C	2071	35	2106	10B57	117B0	3161	35	3196

Table 15: Test 4 LLC Miss Count Values

Test 4	L3_LAT_CACHE.MISS

Core 0	Shared					Partitioned				
	Start Value	Stop Value	Difference	Delta	Sum	Start Value	Stop Value	Difference	Delta	Sum
	B	2A1A	10767		10767	9	2A70	10855		10855
	2A41	3550	2831	39	2870	2A96	36F8	3170	38	3208
	3575	3D8E	2073	37	2110	371B	4374	3161	35	3196
	3DB3	45CC	2073	37	2110	4397	4FF0	3161	35	3196
	45F1	4E0A	2073	37	2110	5013	5C6C	3161	35	3196
	4E2F	5648	2073	37	2110	5C8F	68E8	3161	35	3196
	566D	5E86	2073	37	2110	690B	7564	3161	35	3196
	5EAB	66C4	2073	37	2110	7587	81E0	3161	35	3196
	66E9	6F02	2073	37	2110	8203	8E5C	3161	35	3196
	6F27	7740	2073	37	2110	8E7F	9AD8	3161	35	3196
	7765	7F7E	2073	37	2110	9AFB	A754	3161	35	3196
	7FA3	87BC	2073	37	2110	A777	B3D0	3161	35	3196
	87E1	8FFA	2073	37	2110	B3F3	C04C	3161	35	3196
	901F	9838	2073	37	2110	C06F	CCC8	3161	35	3196
	985D	A076	2073	37	2110	CCEB	D944	3161	35	3196
	A09B	A8B4	2073	37	2110	D967	E5C0	3161	35	3196
	A8D9	B0F2	2073	37	2110	E5E3	F23C	3161	35	3196
	B117	B930	2073	37	2110	F25F	FEB8	3161	35	3196
	B955	C16E	2073	37	2110	FEDB	10B34	3161	35	3196
	C193	C9AC	2073	37	2110	10B57	117B0	3161	35	3196
Test 4 Core 1	L3_LAT_CACHE.MISS									
	Shared					Partitioned				
	Start Value	Stop Value	Difference	Delta	Sum	Start Value	Stop Value	Difference	Delta	Sum
	A	2A1F	10773		10773	A	2A71	10855		10855
	2A48	3557	2831	41	2872	2A98	36FA	3170	39	3209
	357A	3D93	2073	35	2108	371E	4378	3162	36	3198
	3DB6	45CF	2073	35	2108	439C	4FF6	3162	36	3198
	45F2	4E0B	2073	35	2108	501A	5C74	3162	36	3198
	4E2E	5647	2073	35	2108	5C98	68F2	3162	36	3198
	566A	5E83	2073	35	2108	6916	7570	3162	36	3198
	5EA6	66BF	2073	35	2108	7594	81EE	3162	36	3198
	66E2	6EFB	2073	35	2108	8212	8E6C	3162	36	3198
	6F1E	7737	2073	35	2108	8E90	9AEA	3162	36	3198
	775A	7F73	2073	35	2108	9B0E	A768	3162	36	3198
	7F96	87AF	2073	35	2108	A78C	B3E6	3162	36	3198
	87D2	8FEB	2073	35	2108	B40A	C064	3162	36	3198
	900E	9827	2073	35	2108	C088	CCE2	3162	36	3198
	984A	A063	2073	35	2108	CD06	D960	3162	36	3198
	A086	A89F	2073	35	2108	D984	E5DE	3162	36	3198
	A8C2	B0DB	2073	35	2108	E602	F25C	3162	36	3198

	B0FE	B917	2073	35	2108	F280	FEDA	3162	36	3198
	B93A	C153	2073	35	2108	FEFE	10B58	3162	36	3198
	C176	C98F	2073	35	2108	10B7C	117D6	3162	36	3198

Table 16: Test 4 Averages and Comparison to Test 3

	L3_LAT_CACHE.MISS							
	Shared				Partitioned			
	Difference	Sum	Compared Difference	Compared Sum	Difference	Sum	Compared Difference	Compared Sum
Test 4 Average	10770	10770	5	5	10855	10855	0	0
	2831	2871	2	5	3170	3208.5	0	0.5
	2073	2109	2	3	3161.5	3197	0.5	1
	2073	2109	2	3	3161.5	3197	0.5	1
	2073	2109	2	3	3161.5	3197	0.5	1
	2073	2109	2	3	3161.5	3197	0.5	1
	2073	2109	2	3	3161.5	3197	0.5	1
	2073	2109	2	3	3161.5	3197	0.5	1
	2073	2109	2	3	3161.5	3197	0.5	1
	2073	2109	2	3	3161.5	3197	0.5	1
	2073	2109	2	3	3161.5	3197	0.5	1
	2073	2109	2	3	3161.5	3197	0.5	1
	2073	2109	2	3	3161.5	3197	0.5	1
	2073	2109	2	3	3161.5	3197	0.5	1
	2073	2109	2	3	3161.5	3197	0.5	1
	2073	2109	2	3	3161.5	3197	0.5	1
	2073	2109	2	3	3161.5	3197	0.5	1
	2073	2109	2	3	3161.5	3197	0.5	1
	2073	2109	2	3	3161.5	3197	0.5	1
	2073	2109	2	3	3161.5	3197	0.5	1

Table 17: Test 5 LLC Miss Count Values

	L3_LAT_CACHE.MISS									
	Shared					Partitioned				
	Start Value	Stop Value	Difference	Delta	Sum	Start Value	Stop Value	Difference	Delta	Sum
Test 5 Core 0	9	3215	12812		12812	9	2A70	10855		10855
	323A	3D49	2831	37	2868	2A95	36F6	3169	37	3206
	3D6C	4585	2073	35	2108	3718	4372	3162	34	3196
	45A8	4DC1	2073	35	2108	4394	4FEE	3162	34	3196
	4DE4	55FD	2073	35	2108	5010	5C6A	3162	34	3196
	5620	5E39	2073	35	2108	5C8C	68E6	3162	34	3196

	5E5C	6675	2073	35	2108	6908	7562	3162	34	3196
	6698	6EB1	2073	35	2108	7584	81DE	3162	34	3196
	6ED4	76ED	2073	35	2108	8200	8E5A	3162	34	3196
	7710	7F29	2073	35	2108	8E7C	9AD6	3162	34	3196
	7F4C	8765	2073	35	2108	9AF8	A752	3162	34	3196
	8788	8FA1	2073	35	2108	A774	B3CE	3162	34	3196
	8FC4	97DD	2073	35	2108	B3F0	C04A	3162	34	3196
	9800	A019	2073	35	2108	C06C	CCC6	3162	34	3196
	A03C	A855	2073	35	2108	CCE8	D942	3162	34	3196
	A878	B091	2073	35	2108	D964	E5BE	3162	34	3196
	B0B4	B8CD	2073	35	2108	E5E0	F23A	3162	34	3196
	B8F0	C109	2073	35	2108	F25C	FEB6	3162	34	3196
	C12C	C945	2073	35	2108	FED8	10B32	3162	34	3196
	C968	D181	2073	35	2108	10B54	117AE	3162	34	3196
Test 5 Core 1	L3_LAT_CACHE.MISS									
	Shared					Partitioned				
	Start Value	Stop Value	Difference	Delta	Sum	Start Value	Stop Value	Difference	Delta	Sum
	9	341B	13330		13330	A	2A71	10855		10855
	3444	4153	3343	41	3384	2A98	36F9	3169	39	3208
	4175	4B8E	2585	34	2619	371D	4377	3162	36	3198
	4BB0	55C9	2585	34	2619	439B	4FF5	3162	36	3198
	55EB	6004	2585	34	2619	5019	5C73	3162	36	3198
	6026	6A3F	2585	34	2619	5C97	68F1	3162	36	3198
	6A61	747A	2585	34	2619	6915	756F	3162	36	3198
	749C	7EB5	2585	34	2619	7593	81ED	3162	36	3198
	7ED7	88F0	2585	34	2619	8211	8E6B	3162	36	3198
	8912	932B	2585	34	2619	8E8F	9AE9	3162	36	3198
	934D	9D66	2585	34	2619	9B0D	A767	3162	36	3198
	9D88	A7A1	2585	34	2619	A78B	B3E5	3162	36	3198
	A7C3	B1DC	2585	34	2619	B409	C063	3162	36	3198
	B1FE	BC17	2585	34	2619	C087	CCE1	3162	36	3198
	BC39	C652	2585	34	2619	CD05	D95F	3162	36	3198
	C674	D08D	2585	34	2619	D983	E5DD	3162	36	3198
	D0AF	DAC8	2585	34	2619	E601	F25B	3162	36	3198
	DAEA	E503	2585	34	2619	F27F	FED9	3162	36	3198
	E525	EF3E	2585	34	2619	FEFD	10B57	3162	36	3198
	EF60	F979	2585	34	2619	10B7B	117D5	3162	36	3198
Test 5 Core 2	L3_LAT_CACHE.MISS									
	Shared					Partitioned				
	Start Value	Stop Value	Difference	Delta	Sum	Start Value	Stop Value	Difference	Delta	Sum
	9	2C1C	11283		11283	A	2A82	10872		10872
	2C45	3954	3343	41	3384	2AA8	3717	3183	38	3221

	3977	4390	2585	35	2620	373A	43A3	3177	35	3212
	43B3	4DCC	2585	35	2620	43C6	502F	3177	35	3212
	4DEF	5808	2585	35	2620	5052	5CBB	3177	35	3212
	582B	6244	2585	35	2620	5CDE	6947	3177	35	3212
	6267	6C80	2585	35	2620	696A	75D3	3177	35	3212
	6CA3	76BC	2585	35	2620	75F6	825F	3177	35	3212
	76DF	80F8	2585	35	2620	8282	8EEB	3177	35	3212
	811B	8B34	2585	35	2620	8F0E	9B77	3177	35	3212
	8B57	9570	2585	35	2620	9B9A	A803	3177	35	3212
	9593	9FAC	2585	35	2620	A826	B48F	3177	35	3212
	9FCF	A9E8	2585	35	2620	B4B2	C11B	3177	35	3212
	AA0B	B424	2585	35	2620	C13E	CDA7	3177	35	3212
	B447	BE60	2585	35	2620	CDCA	DA33	3177	35	3212
	BE83	C89C	2585	35	2620	DA56	E6BF	3177	35	3212
	C8BF	D2D8	2585	35	2620	E6E2	F34B	3177	35	3212
	D2FB	DD14	2585	35	2620	F36E	FFD7	3177	35	3212
	DD37	E750	2585	35	2620	FFFA	10C63	3177	35	3212
	E773	F18C	2585	35	2620	10C86	118EF	3177	35	3212
Test 5 Core 3	L3_LAT_CACHE.MISS									
	Shared					Partitioned				
	Start Value	Stop Value	Difference	Delta	Sum	Start Value	Stop Value	Difference	Delta	Sum
	9	2A1C	10771		10771	9	2C6F	11366		11366
	2A45	3554	2831	41	2872	2C95	3AF5	3680	38	3718
	3577	3D90	2073	35	2108	3B18	4971	3673	35	3708
	3DB3	45CC	2073	35	2108	4994	57ED	3673	35	3708
	45EF	4E08	2073	35	2108	5810	6669	3673	35	3708
	4E2B	5644	2073	35	2108	668C	74E5	3673	35	3708
	5667	5E80	2073	35	2108	7508	8361	3673	35	3708
	5EA3	66BC	2073	35	2108	8384	91DD	3673	35	3708
	66DF	6EF8	2073	35	2108	9200	A059	3673	35	3708
	6F1B	7734	2073	35	2108	A07C	AED5	3673	35	3708
	7757	7F70	2073	35	2108	AEF8	BD51	3673	35	3708
	7F93	87AC	2073	35	2108	BD74	CBCD	3673	35	3708
	87CF	8FE8	2073	35	2108	CBF0	DA49	3673	35	3708
	900B	9824	2073	35	2108	DA6C	E8C5	3673	35	3708
	9847	A060	2073	35	2108	E8E8	F741	3673	35	3708
	A083	A89C	2073	35	2108	F764	105BD	3673	35	3708
	A8BF	B0D8	2073	35	2108	105E0	11439	3673	35	3708
	B0FB	B914	2073	35	2108	1145C	122B5	3673	35	3708
	B937	C150	2073	35	2108	122D8	13131	3673	35	3708
	C173	C98C	2073	35	2108	13154	13FAD	3673	35	3708

Table 18: Test 5 Averages and Comparison to Test 3

	L3_LAT_CACHE.MISS							
	Shared				Partitioned			
	Difference	Sum	Compared Difference	Compared Sum	Difference	Sum	Compared Difference	Compared Sum
Test 5 Average	12049	12049	1284	1284	10987	10987	132	132
	3087	3127	258	261	3300.25	3338.25	130.25	130.25
	2329	2363.75	258	257.75	3293.5	3328.5	132.5	132.5
	2329	2363.75	258	257.75	3293.5	3328.5	132.5	132.5
	2329	2363.75	258	257.75	3293.5	3328.5	132.5	132.5
	2329	2363.75	258	257.75	3293.5	3328.5	132.5	132.5
	2329	2363.75	258	257.75	3293.5	3328.5	132.5	132.5
	2329	2363.75	258	257.75	3293.5	3328.5	132.5	132.5
	2329	2363.75	258	257.75	3293.5	3328.5	132.5	132.5
	2329	2363.75	258	257.75	3293.5	3328.5	132.5	132.5
	2329	2363.75	258	257.75	3293.5	3328.5	132.5	132.5
	2329	2363.75	258	257.75	3293.5	3328.5	132.5	132.5
	2329	2363.75	258	257.75	3293.5	3328.5	132.5	132.5
	2329	2363.75	258	257.75	3293.5	3328.5	132.5	132.5
	2329	2363.75	258	257.75	3293.5	3328.5	132.5	132.5
	2329	2363.75	258	257.75	3293.5	3328.5	132.5	132.5
	2329	2363.75	258	257.75	3293.5	3328.5	132.5	132.5
	2329	2363.75	258	257.75	3293.5	3328.5	132.5	132.5
	2329	2363.75	258	257.75	3293.5	3328.5	132.5	132.5
	2329	2363.75	258	257.75	3293.5	3328.5	132.5	132.5

Table 19: Test 6 LLC Miss Count Values and Comparison to Test 3

	L3_LAT_CACHE.MISS													
	Shared							Partitioned						
	Start Value	Stop Value	Diff	Delta	Sum	Compared Difference	Compared Sum	Start Value	Stop Value	Diff	Delta	Sum	Compared Difference	Compared Sum
Test 6	a	2a1f	10773		10773	8	8	a	2a73	10857		10857	2	2
	2a49	3558	2831	42	2873	2	7	2a9a	36fa	3168	39	3207	-2	-1
	357c	3d95	2073	36	2109	2	3	371e	4378	3162	36	3198	1	2
	3db9	45d2	2073	36	2109	2	3	439c	4ff6	3162	36	3198	1	2
	45f6	4e0f	2073	36	2109	2	3	501a	5c74	3162	36	3198	1	2
	4e33	564c	2073	36	2109	2	3	5c98	68f2	3162	36	3198	1	2
	5670	5e89	2073	36	2109	2	3	6916	7570	3162	36	3198	1	2
	5ead	66c6	2073	36	2109	2	3	7594	81ee	3162	36	3198	1	2
	66ea	6f03	2073	36	2109	2	3	8212	8e6c	3162	36	3198	1	2
	6f27	7740	2073	36	2109	2	3	8e90	9aea	3162	36	3198	1	2

	7764	7f7d	2073	36	2109	2	3	9b0e	a768	3162	36	3198	1	2
	7fa1	87ba	2073	36	2109	2	3	a78c	b3e6	3162	36	3198	1	2
	87de	8ff7	2073	36	2109	2	3	b40a	c064	3162	36	3198	1	2
	901b	9834	2073	36	2109	2	3	c088	cce2	3162	36	3198	1	2
	9858	a071	2073	36	2109	2	3	cd06	d960	3162	36	3198	1	2
	a095	a8ae	2073	36	2109	2	3	d984	e5de	3162	36	3198	1	2
	a8d2	b0eb	2073	36	2109	2	3	e602	f25c	3162	36	3198	1	2
	b10f	b928	2073	36	2109	2	3	f280	feda	3162	36	3198	1	2
	b94c	c165	2073	36	2109	2	3	fefe	10b58	3162	36	3198	1	2
	c189	c9a2	2073	36	2109	2	3	10b7c	117d6	3162	36	3198	1	2

Table 20: Test 6b LLC Miss Count Values

	L3_LAT_CACHE.MISS									
	Shared					Partitioned				
	Start Value	Stop Value	Diff	Delta	Sum	Start Value	Stop Value	Diff	Delta	Sum
Test 6b Core 1	b	2a20	10773		10773	a	2a73	10857		10857
	2a4a	3559	2831	42	2873	2a9a	36fa	3168	39	3207
	357d	3d96	2073	36	2109	371e	4378	3162	36	3198
	3dba	45d3	2073	36	2109	439c	4ff6	3162	36	3198
	45f7	4e10	2073	36	2109	501a	5c74	3162	36	3198
	4e34	564d	2073	36	2109	5c98	68f2	3162	36	3198
	5671	5e8a	2073	36	2109	6916	7570	3162	36	3198
	5eae	66c7	2073	36	2109	7594	81ee	3162	36	3198
	66eb	6f04	2073	36	2109	8212	8e6c	3162	36	3198
	6f28	7741	2073	36	2109	8e90	9aea	3162	36	3198
	7765	7f7e	2073	36	2109	9b0e	a768	3162	36	3198
	7fa2	87bb	2073	36	2109	a78c	b3e6	3162	36	3198
	87df	8ff8	2073	36	2109	b40a	c064	3162	36	3198
	901c	9835	2073	36	2109	c088	cce2	3162	36	3198
	9859	a072	2073	36	2109	cd06	d960	3162	36	3198
	a096	a8af	2073	36	2109	d984	e5de	3162	36	3198
	a8d3	b0ec	2073	36	2109	e602	f25c	3162	36	3198
	b110	b929	2073	36	2109	f280	feda	3162	36	3198
	b94d	c166	2073	36	2109	fefe	10b58	3162	36	3198
	c18a	c9a3	2073	36	2109	10b7c	117d6	3162	36	3198
Test 6b Core 2	L3_LAT_CACHE.MISS									
	Shared					Partitioned				
	Start Value	Stop Value	Diff	Delta	Sum	Start Value	Stop Value	Diff	Delta	Sum
	a	2a1e	10772		10772	9	2a70	10855		10855
	2a48	3558	2832	42	2874	2a97		3169	39	3208
	357c	3d96	2074	36	2110	371c	36f8	3162	36	3198

	3dba	45d4	2074	36	2110	439a	4ff4	3162	36	3198
	45f8	4e12	2074	36	2110	5018	5c72	3162	36	3198
	4e36	5650	2074	36	2110	5c96	68f0	3162	36	3198
	5674	5e8e	2074	36	2110	6914	756e	3162	36	3198
	5eb2	66cc	2074	36	2110	7592	81ec	3162	36	3198
	66f0	6f0a	2074	36	2110	8210	8e6a	3162	36	3198
	6f2e	7748	2074	36	2110	8e8e	9ae8	3162	36	3198
	776c	7f86	2074	36	2110	9b0c	a766	3162	36	3198
	7faa	87c4	2074	36	2110	a78a	b3e4	3162	36	3198
	87e8	9002	2074	36	2110	b408	c062	3162	36	3198
	9026	9840	2074	36	2110	c086	cce0	3162	36	3198
	9864	a07e	2074	36	2110	cd04	d95e	3162	36	3198
	a0a2	a8bc	2074	36	2110	d982	e5dc	3162	36	3198
	a8e0	b0fa	2074	36	2110	e600	f25a	3162	36	3198
	b11e	b938	2074	36	2110	f27e	fed8	3162	36	3198
	b95c	c176	2074	36	2110	fefc	10b56	3162	36	3198
	c19a	c9b4	2074	36	2110	10b7a	117d4	3162	36	3198
Test 6b Core 3	L3_LAT_CACHE.MISS									
	Shared					Partitioned				
	Start Value	Stop Value	Diff	Delta	Sum	Start Value	Stop Value	Diff	Delta	Sum
	9	2a1d	10772		10772	9	2a81	10872		10872
	2a46	3556	2832	41	2873	2aa6	3715	3183	37	3220
	3579	3d93	2074	35	2109	3737	43a0	3177	34	3211
	3db6	45d0	2074	35	2109	43c2	502b	3177	34	3211
	45f3	4e0d	2074	35	2109	504d	5cb6	3177	34	3211
	4e30	564a	2074	35	2109	5cd8	6941	3177	34	3211
	566d	5e87	2074	35	2109	6963	75cc	3177	34	3211
	5eaa	66c4	2074	35	2109	75ee	8257	3177	34	3211
	66e7	6f01	2074	35	2109	8279	8ee2	3177	34	3211
	6f24	773e	2074	35	2109	8f04	9b6d	3177	34	3211
	7761	7f7b	2074	35	2109	9b8f	a7f8	3177	34	3211
	7f9e	87b8	2074	35	2109	a81a	b483	3177	34	3211
	87db	8ff5	2074	35	2109	b4a5	c10e	3177	34	3211
	9018	9832	2074	35	2109	c130	cd99	3177	34	3211
	9855	a06f	2074	35	2109	cdbb	da24	3177	34	3211
	a092	a8ac	2074	35	2109	da46	e6af	3177	34	3211
	a8cf	b0e9	2074	35	2109	e6d1	f33a	3177	34	3211
	b10c	b926	2074	35	2109	f35c	ffc5	3177	34	3211
	b949	c163	2074	35	2109	ffe7	10c50	3177	34	3211
	c186	c9a0	2074	35	2109	10c72	118db	3177	34	3211

Table 21: Test 6b Averages and Comparison to Test 3

	L3_LAT_CACHE.MISS							
	Shared				Partitioned			
	Difference	Sum	Compared Difference	Compared Sum	Difference	Sum	Compared Difference	Compared Sum
Test 6b Average	10772.33	10772.33	7.33	7.33	10861.33	10861.33	6.33	6.33
	2831.67	2873.33	2.67	7.33	3173.33	3211.67	3.33	3.67
	2073.67	2109.33	2.67	3.33	3167.00	3202.33	6.00	6.33
	2073.67	2109.33	2.67	3.33	2101.00	3202.33	6.00	6.33
	2073.67	2109.33	2.67	3.33	3167.00	3202.33	6.00	6.33
	2073.67	2109.33	2.67	3.33	3167.00	3202.33	6.00	6.33
	2073.67	2109.33	2.67	3.33	3167.00	3202.33	6.00	6.33
	2073.67	2109.33	2.67	3.33	3167.00	3202.33	6.00	6.33
	2073.67	2109.33	2.67	3.33	3167.00	3202.33	6.00	6.33
	2073.67	2109.33	2.67	3.33	3167.00	3202.33	6.00	6.33
	2073.67	2109.33	2.67	3.33	3167.00	3202.33	6.00	6.33
	2073.67	2109.33	2.67	3.33	3167.00	3202.33	6.00	6.33
	2073.67	2109.33	2.67	3.33	3167.00	3202.33	6.00	6.33
	2073.67	2109.33	2.67	3.33	3167.00	3202.33	6.00	6.33
	2073.67	2109.33	2.67	3.33	3167.00	3202.33	6.00	6.33
	2073.67	2109.33	2.67	3.33	3167.00	3202.33	6.00	6.33
	2073.67	2109.33	2.67	3.33	3167.00	3202.33	6.00	6.33
	2073.67	2109.33	2.67	3.33	3167.00	3202.33	6.00	6.33
	2073.67	2109.33	2.67	3.33	3167.00	3202.33	6.00	6.33
	2073.67	2109.33	2.67	3.33	3167.00	3202.33	6.00	6.33

Table 22: Test 7 LLC Miss Count Values

	L3_LAT_CACHE.MISS									
	Shared					Partitioned				
	Start Value	Stop Value	Difference	Delta	Sum	Start Value	Stop Value	Difference	Delta	Sum
Test 7 Core 0	9	6418	25615		25615	9	54CC	21699		21699
	643D	7C4A	6157	37	6194	54F1	6DAD	6332	37	6369
	7C6D	9476	6153	35	6188	6DD0	867B	6315	35	6350
	9499	ACA2	6153	35	6188	869E	9F48	6314	35	6349
	ACC5	C4CE	6153	35	6188	9F6A	B814	6314	34	6348
	C4F1	DCFA	6153	35	6188	B836	D0E0	6314	34	6348
	DD1D	F526	6153	35	6188	D102	E9AC	6314	34	6348
	F549	10D52	6153	35	6188	E9CE	10278	6314	34	6348
	10D75	1257E	6153	35	6188	1029A	11B44	6314	34	6348
	125A1	13DAA	6153	35	6188	11B66	13410	6314	34	6348
	13DCD	155D6	6153	35	6188	13432	14CDC	6314	34	6348

	155F9	16E02	6153	35	6188	14CFE	165A8	6314	34	6348
	16E25	1862E	6153	35	6188	165CA	17E74	6314	34	6348
	18651	19E5A	6153	35	6188	17E96	19740	6314	34	6348
	19E7D	1B686	6153	35	6188	19762	1B00C	6314	34	6348
	1B6A9	1CEB2	6153	35	6188	1B02E	1C8D8	6314	34	6348
	1CED5	1E6DE	6153	35	6188	1C8FA	1E1A4	6314	34	6348
	1E701	1FF0A	6153	35	6188	1E1C6	1FA70	6314	34	6348
	1FF2D	21736	6153	35	6188	1FA92	2133C	6314	34	6348
	21759	22F62	6153	35	6188	2135E	22C08	6314	34	6348
Test 7 Core 1	L3_LAT_CACHE.MISS									
	Shared					Partitioned				
	Start Value	Stop Value	Difference	Delta	Sum	Start Value	Stop Value	Difference	Delta	Sum
	9	681E	26645		26645	A	54CD	21699		21699
	6847	8454	7181	41	7222	54F4	6DB1	6333	39	6372
	8476	A07F	7177	34	7211	6DD6	8681	6315	37	6352
	A0A1	BCAA	7177	34	7211	86A6	9F50	6314	37	6351
	BCCC	D8D5	7177	34	7211	9F74	B81E	6314	36	6350
	D8F7	F500	7177	34	7211	B842	D0EC	6314	36	6350
	F522	1112B	7177	34	7211	D110	E9BA	6314	36	6350
	1114D	12D56	7177	34	7211	E9DE	10288	6314	36	6350
	12D78	14981	7177	34	7211	102AC	11B56	6314	36	6350
	149A3	165AC	7177	34	7211	11B7A	13424	6314	36	6350
	165CE	181D7	7177	34	7211	13448	14CF2	6314	36	6350
	181F9	19E02	7177	34	7211	14D16	165C0	6314	36	6350
	19E24	1BA2D	7177	34	7211	165E4	17E8E	6314	36	6350
	1BA4F	1D658	7177	34	7211	17EB2	1975C	6314	36	6350
	1D67A	1F283	7177	34	7211	19780	1B02A	6314	36	6350
	1F2A5	20EAE	7177	34	7211	1B04E	1C8F8	6314	36	6350
	20ED0	22AD9	7177	34	7211	1C91C	1E1C6	6314	36	6350
	22AFB	24704	7177	34	7211	1E1EA	1FA94	6314	36	6350
	24726	2632F	7177	34	7211	1FAB8	21362	6314	36	6350
	26351	27F5A	7177	34	7211	21386	22C30	6314	36	6350
Test 7 Core 2	L3_LAT_CACHE.MISS									
	Shared					Partitioned				
	Start Value	Stop Value	Difference	Delta	Sum	Start Value	Stop Value	Difference	Delta	Sum
	9	581F	22550		22550	A	54EE	21732		21732
	5848	7455	7181	41	7222	5514	6DEE	6362	38	6400
	7478	9081	7177	35	7212	6E11	86DC	6347	35	6382
	90A4	ACAD	7177	35	7212	86FF	9FCA	6347	35	6382
	ACD0	C8D9	7177	35	7212	9FED	B8B8	6347	35	6382
	C8FC	E505	7177	35	7212	B8DB	D1A6	6347	35	6382
	E528	10131	7177	35	7212	D1C9	EA94	6347	35	6382

	10154	11D5D	7177	35	7212	EAB7	10382	6347	35	6382
	11D80	13989	7177	35	7212	103A5	11C70	6347	35	6382
	139AC	155B5	7177	35	7212	11C93	1355E	6347	35	6382
	155D8	171E1	7177	35	7212	13581	14E4C	6347	35	6382
	17204	18E0D	7177	35	7212	14E6F	1673A	6347	35	6382
	18E30	1AA39	7177	35	7212	1675D	18028	6347	35	6382
	1AA5C	1C665	7177	35	7212	1804B	19916	6347	35	6382
	1C688	1E291	7177	35	7212	19939	1B204	6347	35	6382
	1E2B4	1FEBD	7177	35	7212	1B227	1CAF2	6347	35	6382
	1FEE0	21AE9	7177	35	7212	1CB15	1E3E0	6347	35	6382
	21B0C	23715	7177	35	7212	1E403	1FCCE	6347	35	6382
	23738	25341	7177	35	7212	1FCF1	215BC	6347	35	6382
	25364	26F6D	7177	35	7212	215DF	22EAA	6347	35	6382
Test 7 Core 3	L3_LAT_CACHE.MISS									
	Shared					Partitioned				
	Start Value	Stop Value	Difference	Delta	Sum	Start Value	Stop Value	Difference	Delta	Sum
	9	541F	21526		21526	9	58CB	22722		22722
	5448	6C55	6157	41	6198	58F1	75AD	7356	38	7394
	6C78	8481	6153	35	6188	75D1	927B	7338	36	7374
	84A4	9CAD	6153	35	6188	929F	AF48	7337	36	7373
	9CD0	B4D9	6153	35	6188	AF6B	CC14	7337	35	7372
	B4FC	CD05	6153	35	6188	CC37	E8E0	7337	35	7372
	CD28	E531	6153	35	6188	E903	105AC	7337	35	7372
	E554	FD5D	6153	35	6188	105CF	12278	7337	35	7372
	FD80	11589	6153	35	6188	1229B	13F44	7337	35	7372
	115AC	12DB5	6153	35	6188	13F67	15C10	7337	35	7372
	12DD8	145E1	6153	35	6188	15C33	178DC	7337	35	7372
	14604	15E0D	6153	35	6188	178FF	195A8	7337	35	7372
	15E30	17639	6153	35	6188	195CB	1B274	7337	35	7372
	1765C	18E65	6153	35	6188	1B297	1CF40	7337	35	7372
	18E88	1A691	6153	35	6188	1CF63	1EC0C	7337	35	7372
	1A6B4	1BEBD	6153	35	6188	1EC2F	208D8	7337	35	7372
	1BEE0	1D6E9	6153	35	6188	208FB	225A4	7337	35	7372
	1D70C	1EF15	6153	35	6188	225C7	24270	7337	35	7372
	1EF38	20741	6153	35	6188	24293	25F3C	7337	35	7372
	20764	21F6D	6153	35	6188	25F5F	27C08	7337	35	7372

Table 23: Test 7 Averages and Comparison to Test 3

Test 7 Average	L3_LAT_CACHE.MISS	
	Shared	Partitioned

	Difference	Sum	Compared Difference	Compared Sum	Difference	Sum	Compared Difference	Compared Sum
	12042	12042	1277	1277	10981.5	10981.5	126.5	126.5
	3334.5	3354.5	505.5	488.5	3297.875	3316.875	127.875	108.875
	3332.5	3349.875	1261.5	1243.875	3289.375	3307.25	128.375	111.25
	3332.5	3349.875	1261.5	1243.875	3289	3306.875	128	110.875
	3332.5	3349.875	1261.5	1243.875	3289	3306.5	128	110.5
	3332.5	3349.875	1261.5	1243.875	3289	3306.5	128	110.5
	3332.5	3349.875	1261.5	1243.875	3289	3306.5	128	110.5
	3332.5	3349.875	1261.5	1243.875	3289	3306.5	128	110.5
	3332.5	3349.875	1261.5	1243.875	3289	3306.5	128	110.5
	3332.5	3349.875	1261.5	1243.875	3289	3306.5	128	110.5
	3332.5	3349.875	1261.5	1243.875	3289	3306.5	128	110.5
	3332.5	3349.875	1261.5	1243.875	3289	3306.5	128	110.5
	3332.5	3349.875	1261.5	1243.875	3289	3306.5	128	110.5
	3332.5	3349.875	1261.5	1243.875	3289	3306.5	128	110.5
	3332.5	3349.875	1261.5	1243.875	3289	3306.5	128	110.5
	3332.5	3349.875	1261.5	1243.875	3289	3306.5	128	110.5
	3332.5	3349.875	1261.5	1243.875	3289	3306.5	128	110.5
	3332.5	3349.875	1261.5	1243.875	3289	3306.5	128	110.5
	3332.5	3349.875	1261.5	1243.875	3289	3306.5	128	110.5
	3332.5	3349.875	1261.5	1243.875	3289	3306.5	128	110.5

Table 24: Test 8 LLC Miss Count Values

	L3_LAT_CACHE.MISS									
	Shared					Partitioned				
	Start Value	Stop Value	Difference	Delta	Sum	Start Value	Stop Value	Difference	Delta	Sum
Test 8 Core 0	9	C81E	51221		51221	9	A984	43387		43387
	C844	F855	12305	38	12343	A9A9	DB17	12654	37	12691
	F878	12881	12297	35	12332	DB39	10C83	12618	34	12652
	128A4	158AD	12297	35	12332	10CA5	13DEF	12618	34	12652
	158D0	188D9	12297	35	12332	13E11	16F5B	12618	34	12652
	188FC	1B905	12297	35	12332	16F7D	1A0C7	12618	34	12652
	1B928	1E931	12297	35	12332	1A0E9	1D233	12618	34	12652
	1E954	2195D	12297	35	12332	1D255	2039F	12618	34	12652
	21980	24989	12297	35	12332	203C1	2350B	12618	34	12652
	249AC	279B5	12297	35	12332	2352D	26677	12618	34	12652
	279D8	2A9E1	12297	35	12332	26699	297E3	12618	34	12652
	2AA04	2DA0D	12297	35	12332	29805	2C94F	12618	34	12652
	2DA30	30A39	12297	35	12332	2C971	2FABB	12618	34	12652
	30A5C	33A65	12297	35	12332	2FADD	32C27	12618	34	12652

	33A88	36A91	12297	35	12332	32C49	35D93	12618	34	12652
	36AB4	39ABD	12297	35	12332	35DB5	38EFF	12618	34	12652
	39AE0	3CAE9	12297	35	12332	38F21	3C06B	12618	34	12652
	3CB0C	3FB15	12297	35	12332	3C08D	3F1D7	12618	34	12652
	3FB38	42B41	12297	35	12332	3F1F9	42343	12618	34	12652
	42B64	45B6D	12297	35	12332	42365	454AF	12618	34	12652
Test 8 Core 1	L3_LAT_CACHE.MISS									
	Shared					Partitioned				
	Start Value	Stop Value	Difference	Delta	Sum	Start Value	Stop Value	Difference	Delta	Sum
	9	D024	53275		53275	A	A985	43387		43387
	D04D	10860	14355	41	14396	A9AC	DB21	12661	39	12700
	10882	1408C	14346	34	14380	DB46	10C91	12619	37	12656
	140AE	178B7	14345	34	14379	10CB6	13E00	12618	37	12655
	178D9	1B0E2	14345	34	14379	13E24	16F6E	12618	36	12654
	1B104	1E90D	14345	34	14379	16F92	1A0DC	12618	36	12654
	1E92F	22138	14345	34	14379	1A100	1D24A	12618	36	12654
	2215A	25963	14345	34	14379	1D26E	203B8	12618	36	12654
	25985	2918E	14345	34	14379	203DC	23526	12618	36	12654
	291B0	2C9B9	14345	34	14379	2354A	26694	12618	36	12654
	2C9DB	301E4	14345	34	14379	266B8	29802	12618	36	12654
	30206	33A0F	14345	34	14379	29826	2C970	12618	36	12654
	33A31	3723A	14345	34	14379	2C994	2FADE	12618	36	12654
	3725C	3AA65	14345	34	14379	2FB02	32C4C	12618	36	12654
	3AA87	3E290	14345	34	14379	32C70	35DBA	12618	36	12654
	3E2B2	41ABB	14345	34	14379	35DDE	38F28	12618	36	12654
	41ADD	452E6	14345	34	14379	38F4C	3C096	12618	36	12654
	45308	48B11	14345	34	14379	3C0BA	3F204	12618	36	12654
	48B33	4C33C	14345	34	14379	3F228	42372	12618	36	12654
	4C35E	4FB67	14345	34	14379	42396	454E0	12618	36	12654
Test 8 Core 2	L3_LAT_CACHE.MISS									
	Shared					Partitioned				
	Start Value	Stop Value	Difference	Delta	Sum	Start Value	Stop Value	Difference	Delta	Sum
	9	B025	45084		45084	A	A9C6	43452		43452
	B04E	E85E	14352	41	14393	A9ED	DB9B	12718	39	12757
	E881	1208A	14345	35	14380	DBBE	10D49	12683	35	12718
	120AD	158B6	14345	35	14380	10D6C	13EF7	12683	35	12718
	158D9	190E2	14345	35	14380	13F1A	170A5	12683	35	12718
	19105	1C90E	14345	35	14380	170C8	1A253	12683	35	12718
	1C931	2013A	14345	35	14380	1A276	1D401	12683	35	12718
	2015D	23966	14345	35	14380	1D424	205AF	12683	35	12718
	23989	27192	14345	35	14380	205D2	2375D	12683	35	12718
	271B5	2A9BE	14345	35	14380	23780	2690B	12683	35	12718

	2A9E1	2E1EA	14345	35	14380	2692E	29AB9	12683	35	12718
	2E20D	31A16	14345	35	14380	29ADC	2CC67	12683	35	12718
	31A39	35242	14345	35	14380	2CC8A	2FE15	12683	35	12718
	35265	38A6E	14345	35	14380	2FE38	32FC3	12683	35	12718
	38A91	3C29A	14345	35	14380	32FE6	36171	12683	35	12718
	3C2BD	3FAC6	14345	35	14380	36194	3931F	12683	35	12718
	3FAE9	432F2	14345	35	14380	39342	3C4CD	12683	35	12718
	43315	46B1E	14345	35	14380	3C4F0	3F67B	12683	35	12718
	46B41	4A34A	14345	35	14380	3F69E	42829	12683	35	12718
	4A36D	4DB76	14345	35	14380	4284C	459D7	12683	35	12718
Test 8 Core 3	L3_LAT_CACHE.MISS									
	Shared					Partitioned				
	Start Value	Stop Value	Difference	Delta	Sum	Start Value	Stop Value	Difference	Delta	Sum
	9	A825	43036		43036	9	B183	45434		45434
	A84E	D85E	12304	41	12345	B1A9	EB1D	14708	38	14746
	D881	1088A	12297	35	12332	EB41	1248B	14666	36	14702
	108AD	138B6	12297	35	12332	124AF	15DF8	14665	36	14701
	138D9	168E2	12297	35	12332	15E1B	19764	14665	35	14700
	16905	1990E	12297	35	12332	19787	1D0D0	14665	35	14700
	19931	1C93A	12297	35	12332	1D0F3	20A3C	14665	35	14700
	1C95D	1F966	12297	35	12332	20A5F	243A8	14665	35	14700
	1F989	22992	12297	35	12332	243CB	27D14	14665	35	14700
	229B5	259BE	12297	35	12332	27D37	2B680	14665	35	14700
	259E1	289EA	12297	35	12332	2B6A3	2EFEC	14665	35	14700
	28A0D	2BA16	12297	35	12332	2F00F	32958	14665	35	14700
	2BA39	2EA42	12297	35	12332	3297B	362C4	14665	35	14700
	2EA65	31A6E	12297	35	12332	362E7	39C30	14665	35	14700
	31A91	34A9A	12297	35	12332	39C53	3D59C	14665	35	14700
	34ABD	37AC6	12297	35	12332	3D5BF	40F08	14665	35	14700
	37AE9	3AAF2	12297	35	12332	40F2B	44874	14665	35	14700
	3AB15	3DB1E	12297	35	12332	44897	481E0	14665	35	14700
	3DB41	40B4A	12297	35	12332	48203	4BB4C	14665	35	14700
	40B6D	43B76	12297	35	12332	4BB6F	4F4B8	14665	35	14700

Table 25: Test 8 Average and Comparison to Test 3

Test 8 Average	L3_LAT_CACHE.MISS							
	Shared				Partitioned			
	Difference	Sum	Compared Difference	Compared Sum	Difference	Sum	Compared Difference	Compared Sum
	12038.5	12038.5	1273.5	1273.5	10978.75	10978.75	123.75	123.75
	3332.25	3342.3125	503.25	476.3125	3296.3125	3305.875	126.3125	97.875

	3330.3125	3339	1259.3125	1233	3286.625	3295.5	125.625	99.5
	3330.25	3338.9375	1259.25	1232.9375	3286.5	3295.375	125.5	99.375
	3330.25	3338.9375	1259.25	1232.9375	3286.5	3295.25	125.5	99.25
	3330.25	3338.9375	1259.25	1232.9375	3286.5	3295.25	125.5	99.25
	3330.25	3338.9375	1259.25	1232.9375	3286.5	3295.25	125.5	99.25
	3330.25	3338.9375	1259.25	1232.9375	3286.5	3295.25	125.5	99.25
	3330.25	3338.9375	1259.25	1232.9375	3286.5	3295.25	125.5	99.25
	3330.25	3338.9375	1259.25	1232.9375	3286.5	3295.25	125.5	99.25
	3330.25	3338.9375	1259.25	1232.9375	3286.5	3295.25	125.5	99.25
	3330.25	3338.9375	1259.25	1232.9375	3286.5	3295.25	125.5	99.25
	3330.25	3338.9375	1259.25	1232.9375	3286.5	3295.25	125.5	99.25
	3330.25	3338.9375	1259.25	1232.9375	3286.5	3295.25	125.5	99.25
	3330.25	3338.9375	1259.25	1232.9375	3286.5	3295.25	125.5	99.25
	3330.25	3338.9375	1259.25	1232.9375	3286.5	3295.25	125.5	99.25
	3330.25	3338.9375	1259.25	1232.9375	3286.5	3295.25	125.5	99.25
	3330.25	3338.9375	1259.25	1232.9375	3286.5	3295.25	125.5	99.25
	3330.25	3338.9375	1259.25	1232.9375	3286.5	3295.25	125.5	99.25
	3330.25	3338.9375	1259.25	1232.9375	3286.5	3295.25	125.5	99.25

Table 26: Test 4, 5, 7, and 8 LLC Miss Increase Values

Test 4		Test 5		Test 7		Test 8	
Shared	Partitioned	Shared	Partitioned	Shared	Partitioned	Shared	Partitioned
3	0	257.75	130.25	488.5	108.875	476.3125	97.875
3	0.5	257.75	132	1243.875	110.5	1232.9375	99.25
3	1	257.75	132.5	1243.875	110.5	1232.9375	99.25
3	1	257.75	132.5	1243.875	110.5	1232.9375	99.25
3	1	257.75	132.5	1243.875	110.5	1232.9375	99.25
3	1	257.75	132.5	1243.875	110.5	1232.9375	99.25
3	1	257.75	132.5	1243.875	110.5	1232.9375	99.25
3	1	257.75	132.5	1243.875	110.5	1232.9375	99.25
3	1	257.75	132.5	1243.875	110.5	1232.9375	99.25
3	1	257.75	132.5	1243.875	110.5	1232.9375	99.25
3	1	257.75	132.5	1243.875	110.5	1232.9375	99.25
3	1	257.75	132.5	1243.875	110.5	1232.9375	99.25
3	1	257.75	132.5	1243.875	110.5	1232.9375	99.25
3	1	257.75	132.5	1243.875	110.5	1232.9375	99.25
3	1	257.75	132.5	1243.875	110.5	1232.9375	99.25
3	1	257.75	132.5	1243.875	110.5	1232.9375	99.25
3	1	257.75	132.5	1243.875	110.5	1232.9375	99.25
3	1	257.75	132.5	1243.875	110.875	1232.9375	99.375
5	1	261	132.5	1243.875	111.25	1233	99.5
5	1	1284	132.5	1277	126.5	1273.5	123.75

Table 27: LLC Miss Increase Averages for Tests 4, 5, 7, & 8

	Shared	Partitioned
2MB Dual Core	3.11	0.97
2MB Quad Core	257.93	132.47
4MB Quad Core	1243.88	110.56
8MB Quad Core	1232.94	99.27

LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

ACPI	Advanced Configuration and Power Interface
AES	Advanced Encryption Standard
APIC	Advanced Programmable Interrupt Controller
AP1/AP2/AP3	Application Processor 1/2/3
BIOS	Basic Input Output System
BSP	Bootstrap Processor
CBC	Cipher Block Chaining
CCX	Cache Crossbar
COTS	Commercial Off The Shelf
CPUID	Central Processing Unit Identification
CR0/CR1/CR2/CR3/CR4	Control Register 0/1/2/3/4
DRAM	Dynamic Random Access Memory
EDK	Embedded Design Kit
EPT	Extended Page Table
FPGA	Field Programmable Gate Array
GCC	GNU Compiler Collection
GNU	GNU's Not Unix
GRUB	Grand Unified Bootloader
HDL	Hardware Description Language
IAVMM	Intel-based Hardware-assisted Virtual Machine Monitor
I/O	Input/Output
ISA	Instruction Set Architecture
ISE	Integrated Software Environment
L1/L2/L3	Level 1/2/3
LLC	Last Level Cache
LOC	Lines of code
MILS	Multiple Levels of Security
MMP	Multi-core MILS Processor
MSL	Multiple Singe Layer
MSR	Memory Specific Register

MTRR	Memory Type Range Registers
NCID	Non-inclusive Cache Inclusive Directory
OS	Operating System
RAM	Random Access Memory
RFO	Request for Ownership
SK	Separation Kernel
SMT	Simultaneous Multithreading
TVMM	Tiny Virtual Machine Monitor
UI	University of Idaho
VM	Virtual Machine
VMCS	Virtual Machine Control Structure
VMM	Virtual Machine Monitor
VMX	Virtual Machine Extensions
XUP	Xilinx University Program